

Knowledge-Based Process Planning for Construction and Manufacturing

Knowledge-Based Process Planning for Construction and Manufacturing

Carlos Zozaya-Gorostiza
Chris Hendrickson
Daniel R. Rehak

*Department of Civil Engineering and
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, Pennsylvania*



ACADEMIC PRESS, INC.
Harcourt Brace Jovanovich, Publishers
Boston San Diego New York
Berkeley London Sydney
Tokyo Toronto

658.503
Z91k
C.2

Copyright © 1989 by Academic Press, Inc.

All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording, or any information storage and retrieval system, without permission in writing from the publisher.

ACADEMIC PRESS, INC.

1250 Sixth Avenue, San Diego, CA 92101

United Kingdom Edition published by

ACADEMIC PRESS INC. (LONDON) LTD.

24-28 Oval Road, London NW1 7DX

IBM, PC/AT, and RT PC are registered trademarks of International Business Machines.

KNOWLEDGE CRAFT is a trademark of Carnegie Group, Inc.

LOTUS and 1-2-3 are registered trademarks of Lotus Development Corporation.

MacDraw is a trademark of Claris, Inc.

SCRIBE is a registered trademark of Scribe Systems.

Sun Workstation is a registered trademark of Sun Microsystems, Inc.

EXPLORER is a trademark of Texas Instruments.

Library of Congress Cataloging-in-Publication Data

Zozaya-Gorostiza, Carlos, Date-

Knowledge-based planning for construction and manufacturing/

Carlos Zozaya-Gorostiza, Chris Hendrickson, Daniel R. Rehak.

p. cm.

Bibliography: p.

Includes index.

ISBN 0-12-781900-2

1. Industrial project management—Data processing. 2. Production planning—Data processing. 3. PLANEX (Computer program)

I. Hendrickson, Chris. II. Rehak, Daniel R. III. Title.

TA190.Z69 1989

89-6789

658.5'03'0285—dc20

CIP

Printed in the United States of America

89 90 91 92 9 8 7 6 5 4 3 2 1

Contents

<i>Preface</i>	vii
1 Introduction	1
1.1 Approaches to Process Planning	3
1.2 Motivation for Automated Process Planning	4
1.3 Knowledge-Based Methodology	6
1.4 PLANEX: A Knowledge-Based System for Process Planning	8
1.5 Development and Use of PLANEX	10
2 Process Planning and Scheduling	13
2.1 Plan Formulation with AI Planners	14
2.2 Project Scheduling with Optimization Methods	35
2.3 Construction and Manufacturing Planning	48
2.4 Conclusions	62
3 Modeling Process Planning Problems	65
3.1 A Conceptual Model for Process Planning	66
3.2 Two Illustrative Models for Process Planning Operators	71
3.3 Requirements for a Process Planning Architecture	79
3.4 Conclusions	85
4 A Knowledge-Based Architecture for Process Planning	87
4.1 Overview of PLANEX	88
4.2 Knowledge Representation	91
4.3 Problem Solving and Control	100
4.4 User Interaction	123
4.5 Conclusions	134

5	Developing Process Planning Systems	137
5.1	Capabilities of the PLANEX Architecture	138
5.2	Development of Process Planning Systems Using PLANEX	148
5.3	Example PLANEX Applications	154
5.4	Evaluation of the PLANEX Architecture	170
6	CONSTRUCTION PLANEX: An Expert System for Construction	
	Project Planning	177
6.1	Models for Construction Planning Used in the System	178
6.2	System Architecture	185
6.3	Use of CONSTRUCTION PLANEX	229
6.4	Example Problem	236
6.5	Conclusions	248
7	HARNESS PLANEX: An Expert System for Electrical Wire Harness	
	Process Planning	253
7.1	Models for the Harness Manufacturing Planning Process	254
7.2	System Architecture	258
7.3	Example Problem	280
7.4	Conclusions	288
	<i>References</i>	291
	<i>Index</i>	301

Preface

Construction and manufacturing process planning is a crucial and challenging management task. A good plan is essential to project success. Plans are the basis for a project budget and production schedule. Despite its importance and complexity, process planning relies on manual formulation of plans and is usually performed in an intuitive and unstructured fashion with considerable reliance on engineering judgment.

There are few computer-based process planning aids which address the needs and complexities of the construction and manufacturing processes. Those aids which do exist are primarily algorithmic analysis or graphical display tools. Knowledge-based program development methodologies provide a new technological basis for the development of process planning tools. This approach provides the means to represent and utilize process planning knowledge and judgment which is lacking in current tools. This work describes a knowledge-based system architecture used to develop process planning systems—PLANEX.

PLANEX is a domain-independent, knowledge-based process planning system architecture. Starting from a description of the physical artifact to be constructed or manufactured, PLANEX generates the set of activities used to create the artifact. These activities, with their required resources, are linked into a process planning network which can be used in project scheduling or management. This work presents the concepts, requirements and the resulting architecture of PLANEX, and includes detailed descriptions of applications of the system in construction and manufacturing.

This work originated as an investigation of the application of knowledge-based systems technology in construction management. When the work was conceived in 1984, there were no significant prototype knowledge-based systems in the construction domain; only a few small-scale exploratory applications had been developed. The second and third authors had done extensive work in the application of expert systems in other areas of Civil Engineering, and it was

evident that the technology could be successfully applied to problems which were intractable using conventional programming methodologies.

Creating a project schedule is knowledge-intensive and a prerequisite to the application of other computer tools in project management. Due to the importance of this task, the lack of suitable aids and the promise offered by a knowledge-based systems approach, construction project planning was selected as an appropriate domain to explore in detail. The initial goals were two-fold: (1) to demonstrate the applicability of the knowledge-based approach to the problem; and (2) to investigate the problems with the technology which need to be addressed for it to be successfully applied in this domain.

Although the general domain-independent nature of the process planning problem was considered from the outset, the initial goal was to develop a system limited to construction project planning. In the initial stages, a variety of domains were considered: buildings, bridges and highways. Building construction was selected as a focus and the work commenced. Initial planning explorations were made in the perspicuous *blocks-world* domain, and as the work evolved, it moved from a system targeted at construction to the more general process planning architecture presented herein.

While this work originated in the construction management domain, the ideas and concepts presented have wider application. As presented, the emphasis is on process planning as opposed to project planning. Many of the examples and illustrations are derived from construction planning. This should not be interpreted as a bias in the concepts, but rather as a result of the authors' experience and knowledge.

This work is directed at two audiences: (1) builders and developers of practical process planning aids; and (2) researchers of planning and scheduling systems. It is particularly relevant to those whose research is in the areas of artificial intelligence planning systems; operations research and management science optimization methods for planning; management information and decision support systems; and knowledge-based planning systems.

The development of process planning tools could substantially benefit from generalization and from cross-fertilization among the different planning and decision support approaches. Researchers familiar with only one approach should broaden their horizons and thereby improve their own contributions. Therefore, a major objective of this work was to foster just this sort of cross-fertilization.

Because of the diverse backgrounds and interests of different readers, it is not expected that the individual reader will devote the same attention to each section of this book. For example, operations researchers might omit Section 2.2 reviewing plan scheduling methods. As another example, researchers seeking only a conceptual understanding of knowledge-based process planning might omit the implementation details of the construction and manufacturing applications in Sections 6.2 and 7.2.

Organization This monograph begins with an overview of the process planning problem and the motivation for this work, presented in Chapter 1. This introduction includes a discussion of the knowledge-based approach, a brief overview of PLANEX and a review of its development.

Chapter 2 is a review of background material relevant to this work. It includes a review of AI-based models for process planning and plan formulation; models for deterministic project scheduling, including critical path, resource allocation and resource leveling algorithms; and prior work in the practice of construction and manufacturing planning, scheduling and monitoring. This background material is used to develop a formal model for process planning. The characteristics of this model are described in terms of a conceptual process planning model, and the resulting requirements for a knowledge-based process planning model are presented in Chapter 3. Based on these requirements, the architecture of PLANEX is described in Chapter 4. This description includes the overall structure, knowledge representation, control and user interface components of the architecture.

Use of this system architecture to develop process planning models is presented in Chapter 5. In addition to the basic features of the PLANEX architecture and the process of developing a system with PLANEX, the chapter includes an overview of four different applications (construction planning, excavation planning, manufacturing planning, and blocks-world planning) and an evaluation of the architecture. Chapter 6 provides a detailed description of CONSTRUCTION PLANEX, a system used to plan the construction of mid-rise concrete and steel-frame office buildings. This chapter includes a discussion of the construction planning models used in CONSTRUCTION PLANEX, along with details of the representation, knowledge, problem-solving and user interface components of the system. It concludes with the presentation of an example problem. HARNESS PLANEX, a system which plans the manufacturing operations for automobile electrical wire harnesses, is presented in Chapter 7. The organization and structure of this chapter parallels that of Chapter 6. These two final chapters are intended to provide sufficient detail for implementation of knowledge-based process planning systems so that builders of new tools can use this information as a starting point. More casual readers can skip some sections of these chapters.

PLANEX Software PLANEX has gone through several cycles of development and refinement. The initial implementation was in LISP. The CONSTRUCTION PLANEX and HARNESS PLANEX prototypes described in Chapter 6 and 7 were implemented in COMMON LISP and KNOWLEDGE CRAFT™ on a TEXAS INSTRUMENTS EXPLORER™. Associated subsystems, such as the ANIMATOR, were developed in C on a SILICON GRAPHICS IRIS Workstation. The examples presented herein were developed with this version of PLANEX.

Recently a version of PLANEX was implemented entirely in COMMON LISP, and has been successfully ported to a number of hardware platforms including an IBM RT PC®, an IBM PC/AT® and a Sun Workstation®. This COMMON LISP version does not include graphical schedule displays or the animation subsystem, but generates activity plans for input to commercial scheduling packages.

The COMMON LISP version of PLANEX is available to researchers, educators and institutions who wish to experiment, to refine or to extend the system. Anyone who wishes to obtain PLANEX should contact the second author (CH) at Carnegie Mellon.

Acknowledgments This monograph is based on the Ph.D. dissertation of the first author [115]. The work was supervised by the second author, and was conducted, in part, under a research grant awarded to the second and third authors. In preparing this book, the authors have reorganized, refined and extended the material from the dissertation.

Development of PLANEX was supported in part by the National Science Foundation, Grant MSM-8503400; by the Engineering Design Research Center of Carnegie Mellon University, an NSF Engineering Research Center; by the U.S. Army Construction Engineering Research Laboratory; and by the Department of Civil Engineering of Carnegie Mellon University. Additional support to prepare this monograph was provided by the Engineering Design Research Center.

Computer facilities used to develop PLANEX and to prepare this book were provided by the Engineering Design Research Center, the Department of Civil Engineering and the Robotics Institute of Carnegie Mellon. The authors gratefully acknowledge the funding and financial support which made this work possible.

A number of individuals have made contributions to the development of PLANEX and the applications described herein. Much of the initial construction planning knowledge came from Eduardo Baracco-Miller. Throughout much of the project, Peter S. Lim worked as an undergraduate programmer and was responsible for developing many of the tools and utilities incorporated in the PLANEX architecture. The expertise and skill of these individuals was invaluable in the development effort.

The authors would like to thank two of our colleagues, Prof. Clive L. Dym of the University of Massachusetts and Prof. Raymond E. Levitt of Stanford University for their critical comments in reviewing our manuscript and their continued support and interest throughout the development of PLANEX. We would also like to thank numerous industrial colleagues for the comments reviewing the various prototype systems.

This monograph was produced electronically. Pages were laserset using the SCRIBE® document production system and figures were produced with MacDraw™. Copyediting was done by Heather Walls. The production editor was Joni Hopkins of Academic Press. Sponsoring editor was Sari Kalin of Academic Press.

Carlos Zozaya-Gorostiza
Chris Hendrickson
Daniel R. Rehak

100

100

1 Introduction

This study describes the development of knowledge-based computer tools that assist an engineer in *process planning* (i.e., formulating a *process plan*) for construction and manufacturing. The result of the investigation is a comprehensive knowledge-based system architecture for process planning, called PLANEX, which has been used to develop a number of prototype process planning systems in construction and manufacturing domains.

The issues and problems in process planning can be illustrated through a simple, everyday example: preparing a meal.

- Meal preparation starts with a menu of the dishes which comprise the meal. The menu corresponds to the *specification* of the *product* to be prepared.
- Recipes provide the descriptions of how to create each dish. Depending on the source, the recipes may provide a detailed description of the *tasks* or *activities* used to create the dish, or they may be an *abstract* description of the preparation process.
- No matter how much detail is provided in a recipe, all of the steps for all of the activities are not specified (e.g., recipes often call for *clarified butter* but do not explain how to prepare it). The cook must determine, based on experience, all of the steps needed to prepare the dish (a process denoted *activity formulation*).
- Individual steps from various recipes can often be combined into larger-scale *project activities* (e.g., chopping all vegetables for a dish at one time in a food processor rather than individually). Such aggregation reduces the size of the planning problem.
- Associated with each recipe is the normal yield and the list of ingredients. These *material resources* have to be scaled to provide the desired yield. This is a complex process as some ingredients (e.g., spices, eggs) do not scale linearly, the quantity needed depends upon the form of the ingredient (e.g., fresh versus dried herbs) and complex substitutions for ingredients may be made.

- Steps may be performed by a variety of means (e.g., mixing dough by hand, with a mixer, with a food processor). Given the variety of *technology choices* for each step, the cook must decide what procedures and equipment to use on the basis of skill, experience and knowledge about how the process and technology will affect the preparation and quality of the product.
- Availability of *equipment resources* and competing or conflicting demands for resources (e.g., baking bread requires a hot oven, drying meringues requires a cool oven) place *constraints* on the activities and influence how the cook decides to perform the task.
- A number of *external factors*, such as availability of raw ingredients, influence many decisions made in planning meal preparation.
- The time required to prepare each dish must be estimated. The amount of product needed, type of methods used and skill in performing the individual tasks (i.e., *productivity*) impact this *duration estimation*.
- Recipes also specify *lags* between operations (e.g., dough must rest at least 20 minutes before rolling) or time or *window constraints* (e.g., a step may be done up to one day ahead, but not less than 4 hours ahead).
- Individual recipes indicate some of the *precedences* between the steps used to prepare each item. Precedences from the individual recipes must be combined, with considerations of resource and time constraints, into the complete *activity network* which describes the meal preparation process.
- Given all of the activities and precedences, *scheduling* the activities determines when each should be performed. The schedule must meet a set of *target deadlines* to insure the meal is served on time.

All of these steps constitute the formulation of the process plan, which is uncoupled from the actual preparation of the meal. During meal preparation, a variety of *project management* issues arise, such as *contingency management* (e.g., what to do if one of the steps fails, or if the estimated activity durations are wrong). While not formally considered in this work, management issues and problems parallel the considerations used in developing the initial process plan.

The items listed above represent the components of a *generic plan*. Considerable effort is needed to instantiate this plan for a specific menu, cook and *batterie de cuisine*. Preparing a meal is an everyday task; while many people lack the technical skills required, the planning process is conceptually straightforward. Planning becomes more difficult as the complexity of the dishes increases and as the number of options, *interactions* and *constraints* grow. Construction or manufacturing process planning involves the same issues and requires the same type of problem solving as needed in the cooking domain. The scope, complexity and experience required is substantially larger. Tools to assist the engineer or manager in developing a process plan are essential.

The remainder of this chapter describes process planning and alternative techniques for generating process plans. Then the motivations for developing an

automated process planner and the justifications for using a knowledge-based approach in developing process planning systems are presented. This discussion is followed by an overview of PLANEX and a description of its development.

1.1 Approaches to Process Planning

Given a product to be manufactured or a facility to be constructed, process planning is a fundamental step which is used to map the design of the product onto the methods used to create it. Process planning involves:

- recognition of the elements of the product;
- definition of work tasks used to construct or manufacture each element;
- choice of manufacturing or construction technologies and resources used in these tasks;
- estimation of durations and costs for individual tasks; and
- preparation of project schedules.

The resulting plan consists of the selected resources and technologies associated with the tasks, and the assignment of the tasks to time slots in the schedule.

As detailed below, process planning is important, complex and requires experience. Due to its importance, several attempts have been made to develop computer aids for process planning. While most tools do little to aid in creating a process plan, they are still important to the overall project management process. More comprehensive automated planning systems or planning assistants would be of significant value in construction and manufacturing.

Existing automated process planners can be grouped into two major categories on the basis of the strategy employed to form the process plan [6]:

- *Generative* planners synthesize new plans. Using a description of the product and information describing the basic actions available to create the product, the planner generates a collection of operations and associated resources which together are used to perform parts of the manufacturing or construction process. These individual operations are then ordered into a time sequence of steps used to create the artifact. Each time the planner is invoked it creates a new, unique plan.
- *Retrieval-based* planners select a plan from a library of standard plans. Existing plans are classified according to "key" features of the product and are saved in a plan library. When a new, similarly classified product is defined, the plan which is the "closest" match to the product is retrieved from the library. Plans may be formulated by extracting complete plans from the library or by selecting and combining components from several plans.

An alternative characterization of the planning process is based on considerations of the key elements of the plan and how the planner treats these elements while formulating the plan:

- *Activity-centered* planners are organized around formulating construction and manufacturing activities as the fundamental part of the plan. The activities used to create the product are identified from the design. The technologies and resources used to perform the activities are chosen. The set of activities are organized into a process plan and project schedule. The plan is the set of activities and the associated processes.
- *Work-centered* planners treat the assignment of activities to resources or work centers as the major task in formulating the plan. Sets of activities and schedules are developed for individual work centers and the overall plan is the flow of elements through the work centers. Activities used to create the artifact are based on the capabilities of the work centers. This approach is particularly suited to planning the operation of flexible manufacturing systems.
- *Object-centered* planners consider the artifact or aspects of it as the key to creating the process plan. The approach is similar to the activity centered approach, but aggregations and representations are organized around elements and design objects instead of around the construction and manufacturing activities associated with the objects.

Despite significant efforts to develop automated process planners by using a variety of methodologies and formalisms, effective planning systems which address the needs of the construction and manufacturing communities do not yet exist.

1.2 Motivation for Automated Process Planning

The relevance of this work derives from the needs and issues (described below) that arise in the development of process planning systems. These motivating issues directly lead to consideration of the Artificial Intelligence (AI) based methodology described in the following section.

Process planning is both crucial to and challenging for the successful management of projects. It is crucial to the eventual success of a project because project control and monitoring is based on a particular project plan. Poor estimates or schedules can easily result in cost increases or completion delays. Similar effects may result from inappropriate or inconsistent decisions regarding the resources and technologies selected to perform tasks.

Because the planner is concerned with the formulation of a good plan, rather than just a feasible plan, the planning task is challenging. There are numerous constraints that complicate the planning process, including those related to the availability of resources, completion deadlines for tasks or limitations on project budget. In addition, decisions such as the choice of technology and task decomposition are usually interdependent. The planner has to identify these constraints and interactions and use his experience from previous projects to resolve the resulting problems and conflicts.

Despite the importance and complexity of process planning, little attention has been paid to analyzing the methods by which plans are or should be formed. Planning requires experience and knowledge related to resources, tasks, technologies, budget, schedule and product design. Balancing all of the competing issues and insuring that all aspects are considered is not trivial. Current process planning relies upon manual formulation of plans and is usually performed in an intuitive and unstructured fashion with considerable reliance on engineering judgment. The mechanisms used in planning have not been published and are not formally taught to novices, but rather must be acquired and personalized through experience.

Another pertinent issue is the relationship between plan formulation and planning tools. Few process planning aids exist, and the tools that do exist are better categorized as analysis tools which require an existing plan, rather than tools which aid in plan formation. Most existing computer tools are applicable only to some parts of the whole process. For example, commercial scheduling systems require a complete specification of the project network as input. Such scheduling systems require that decisions concerning plan formulation and refinement be made separately from project scheduling decisions. Once a project network has been input and a schedule computed, the systems provide little support to maintain and refine the schedule while the work is in progress. Developing an integrated computer tool for process planning provides a unified framework for analyzing the interdependencies among planning decisions.

Many of the problems with current process planning tools are related to the inadequacy of the programming methodology used in developing the tools. Commercial construction and project management aids are implemented using *algorithmic* or *procedural* programming. This programming methodology does not provide a convenient mechanism for representing, formalizing or using acquired expertise. Nor do current methods support the development of an integrated process planning environment.

With increased reliance on computer-aided design (CAD) systems for design and computer-aided manufacturing (CAM) systems for manufacturing and computer-integrated construction (CIC) systems for construction, the automated generation of process plans becomes more important in realizing the full potential of the other tools. Integrated tools and comprehensive process planners could provide the ability to reason about construction or manufacturing methods during design, providing better products through *design for manufacturability* or *design for constructability*. Well-designed products and facilities should be relatively easy to manufacture or construct. Current process planning tools and methodologies are inadequate to achieve these goals.

The similarities of process planning in different domains can be exploited in the development of planning aids. For example, there are many parallels between the planning tasks performed by a construction planner and those per-

formed by a process planner in manufacturing. Process planners must identify and sequence the machining operations for manufacturing specific products. Similarly, construction planners have to identify and sequence construction activities for building parts of facilities. Thus, while this work is targeted specifically at construction and manufacturing planning, and has its roots in the construction domain, many of the contributions of this study are applicable to the more general problem of process planning.

Thus, the motivations for developing computer tools for process planning are:

- process planning is crucial in design and project management;
- process planning is a difficult, knowledge-intensive, challenging process;
- there are virtually no integrated computer tools that assist during the complete process planning and project management cycle; and
- there are similarities in process planning across different domains.

1.3 Knowledge-Based Methodology

Given the goals and motivations for developing a process planning system, selecting the appropriate development methodology for building such a system is essential. As prior approaches to solving the problem have not proven successful, the use of a different methodology is indicated. Several AI techniques appear to provide a promising approach for the development of a planning assistant for construction and management.

In particular, the techniques and methodologies of knowledge-based systems¹ are utilized in this investigation. The justifications for selecting this approach are based on the characteristics of knowledge-based systems and the applicability of knowledge-based systems to the process planning problem.

The knowledge-based approach can be characterized and contrasted to traditional algorithmic programming or procedural programming development methodologies by:

- the use of expert, domain-specific knowledge to attain a high level of performance in a narrow domain;
- separation of data, knowledge and control;
- transparency of knowledge representation and dialog (explanation); and
- incremental growth capability.

These characteristics make knowledge-based systems a promising means of representing and using expertise and knowledge in a program. Independent of

¹ The terms *expert system* and *knowledge-based system* are considered interchangeable in this work. However, this work itself is better characterized as knowledge-based as it does not rely only upon expertise.

the availability of expertise, the knowledge-based programming paradigm provides an excellent mechanism for declarative programming, yielding programs that are clearer and more robust.

The planning and management processes are knowledge-intensive, and knowledge-based systems, in addition to providing a mechanism for processing knowledge, are useful in formalizing and structuring the expertise of planners. Since the process planning knowledge of skilled planners is private and idiosyncratic, formalizing the knowledge is beneficial in that it facilitates its refinement and dissemination by organizing and expounding the expertise, making the knowledge available for critical review and analysis.

Successes in the construction industry indicate that the choice of a knowledge-based methodology is appropriate for developing computer tools for project planning and management. Several expert system applications in construction engineering and management have been developed, or are under development [63]. Most of these systems are experimental prototypes that have not yet been used in practice. However, these prototypes have shown that expert and knowledge-based systems are applicable in many areas of the construction industry. As Levitt points out [63, p. 107]:

The extent and breadth of work already completed, under way, or in the early conceptual stages, indicates that many researchers and practitioners in the construction industry see expert systems as offering new and potentially valuable capabilities to support decision-making in the industry.

These efforts are motivated by the desire to improve the efficiency of the construction planning and monitoring processes. They provide tools to assist in solving problems (e.g., scheduling generation, site layout, estimating) for which no tools exist or where algorithmically-based programs are inadequate and ineffective.

A number of prototype knowledge-based systems also have been developed for manufacturing process planning [54, 106]. Here again the tasks are knowledge-intensive and conventional tools have not met the needs of the manufacturing community. In addition, work in the manufacturing domain is motivated by the need to link computer-aided design systems with computer-aided manufacturing systems. Preliminary results indicate that AI-based solutions will succeed in producing capable and effective management and production planning aids.

As noted, there are similarities between construction project planning and manufacturing process planning. Developing knowledge-based systems in one area can be beneficial to system-building efforts in the other areas as the cross-fertilization between the domains can improve the development of process planning tools. Concepts, problems, issues, solutions, etc., from one area lead to a new way of examining, characterizing and solving problems in the other areas.

Because use of the knowledge-based methodology requires critical examination and formulation of knowledge, concepts, solution structure, etc., the potential for synergistic interactions between the different domains is enhanced through the use of the knowledge-based approach.

Thus, the justifications for the application of a knowledge-based methodology in developing process planning systems are:

- process planning is knowledge-intensive;
- knowledge-based systems provide a practical means for representing and using process planning knowledge;
- knowledge-based approaches will yield needed tools which can improve the practice of process planning;
- it has been shown that knowledge-based systems for process planning are feasible; and
- developing knowledge-based tools in one process planning domain (e.g., construction) provides valuable experience and information which can be used in developing expert systems in other process planning domains (e.g., parts manufacturing).

1.4 PLANEX: A Knowledge-Based System for Process Planning

Given the problems with existing tools for process planning, the motivations for developing improved tools, and the justifications for investigating a knowledge-based approach, translating these issues and concepts into an operational system is still difficult. Design and development issues to consider include:

- the scope of the tool (e.g., narrow and specific to one domain or general-purpose and domain-independent);
- the role of the tool (e.g., an autonomous program, a user assistant or an integrated problem-solving environment);
- the problem-solving approach (e.g., a generative planner or one which selects and modifies standard plans);
- the planning strategy (e.g., work-centered planning or activity-centered planning);
- the overall structure and architecture of the system; and
- the validation and maintenance of the system.

These general characteristics along with a myriad of more detailed features characterize the structure and design of a process planning system.

PLANEX is but one alternative for developing a knowledge-based process planning system. It is a generic, domain-independent knowledge-based architecture for process planning in construction and manufacturing. It is a generative planner, and is designed to function as a user assistant.

The general characteristics of the PLANEX architecture can be described in terms of the components of the system and how they are used in process planning.

- All the information relevant to the planning process is stored in the form of *objects*². Stored information includes design components, process activities and resources. The objects are organized into *representational structures*, and an object may be a part of one or more of these structures.
- Process planning knowledge is represented in sets of one or more rules in *Knowledge Sources* (KSs). A knowledge source resembles a decision table and is implemented as a *context object*. Rules in a knowledge source may reference objects and values computed by other knowledge sources. Knowledge representation is uncoupled from the operators which use the knowledge. Knowledge sources are elements of representational structures, permitting knowledge to be hierarchically structured with respect to importance and represented at different levels of abstraction.
- Process planning tasks such as technology choice or activity duration estimation are performed by *domain operators*. Knowledge sources provide the planning knowledge needed by a domain operator. Using representational structures, domain operators can be organized into layers, creating a structure similar to that of MOLGEN (see p. 30). Domain operators are implemented as procedural functions and perform the following steps:
 - Step 1.* Identify the knowledge source to be evaluated. For domain operators which are purely algorithmic, evaluation of a knowledge source is not required.
 - Step 2.* Evaluate the knowledge source using the KNOWLEDGE SOURCE EVALUATOR (KSE). Knowledge source evaluation may be recursive, or may require the evaluation of auxiliary procedures. Evaluation yields a list of results.
 - Step 3.* Store the results in the appropriate context objects.
 - Step 4.* Store the name of the knowledge source which was evaluated.
- Declarative control information describing a domain operator is stored in a *Domain Operator Schema* (DOS). This control information is expressed in terms of the data required (*preconditions*) and the data produced (*effects*) by the operator. Declarative control information provides modularity and permits operators to be unilaterally added or changed.
- An *agenda* schema contains dynamic state information generated during planning. State information used in control includes: (1) the goals the system is

² An object is often called a *schema* and is implemented as a *frame*. In this work, these terms are interchangeable.

trying to achieve; (2) the pending domain operators; (3) precedence information used to sequence the pending operators; and (4) the changes in a context object introduced by executing a domain operator.

- Control of the planning process is provided by four algorithmic *control operators* which use the information in the domain operator schemas and the agenda schema³. Control operators are used to: (1) execute domain operators in an opportunistic manner; and (2) build hierarchical, strategic meta-plans which control the planning process in a manner similar to ABSTRIPS (see p. 20). The four control operators are:
 - the *Forward Propagation Operator* (FPO) which identifies, on the basis of changes in context objects introduced by other domain operators, those domain operators that may be executed;
 - the *Backward Search Operator* (BSO) which finds sequences of domain operators that may be used to achieve a goal;
 - the *Network Interpretation Operator* (NIO) which determines domain operator precedences on the basis of their preconditions and effects; and
 - the *Domain Operator Executor* (DOE) which executes domain operators.
- Overall control is provided through the CONTROL PANEL. The CONTROL PANEL is a user interaction mechanism which provides capabilities to execute a specific control operator or to change the information stored in the agenda.

PLANEX includes components used to create and update knowledge sources, and a set of user interface utilities. As described below, the system is operational and has been used to build several process planners.

1.5 Development and Use of PLANEX

PLANEX, like most complex knowledge-based systems, has gone through several cycles of development and refinement. Development alternated between conceptualization and implementation. The initial work was aimed at developing a specialized system for construction project planning: the generation of work elements, activities, precedences among activities, resource requirements and task durations, coupled with project scheduling. As the work proceeded, it became apparent that the evolving system architecture was applicable to a variety of domains.

Developing a system design without experience and experimentation in developing a knowledge-based application in the construction domain is difficult if not impossible. Thus, an initial prototype limited to excavation planning

³ Goals, object changes, operator preconditions and operator effects are expressed in terms of data *existence*, not in terms of data *values*.

was developed. This first implementation was in LISP on general-purpose workstations. The goal was to explore the necessary concepts and fundamental characteristics of a more complete system for construction project planning.

Based on this first prototype, the system architecture of PLANEX was developed. The first version of CONSTRUCTION PLANEX utilized the concepts and design philosophy of this initial system architecture. This version of CONSTRUCTION PLANEX was capable of planning the excavation and structural erection of low- to mid-rise concrete-framed office buildings. The system was developed in COMMON LISP and KNOWLEDGE CRAFT® on a TEXAS INSTRUMENTS EXPLORER™. It took advantage of the frame-based programming environment provided by KNOWLEDGE CRAFT and the user interface tools provided by the EXPLORER development environment. At this point in the development process, a separate PLANEX architecture did not exist, nor was it fully evident when work on CONSTRUCTION PLANEX began that the evolving system design was applicable to other domains, although the analogies between construction and manufacturing process planning had been considered from the beginning. As the work proceeded, the possibility of refining the system into a domain-independent framework emerged.

The next step in the development process was the refinement and modification of CONSTRUCTION PLANEX to yield the generic, domain-independent process planning model described in Chapter 3 and the knowledge-based system architecture of PLANEX presented in Chapter 4. The implementation environment remained COMMON LISP and KNOWLEDGE CRAFT on a TEXAS INSTRUMENTS EXPLORER. This first complete implementation of the domain-independent PLANEX architecture was used to implement HARNESS PLANEX, which plans the manufacturing of automotive electrical wire harnesses (see Chapter 7); EXCAVATION PLANEX, which plans the excavation of building foundations by robotic excavators; and to reimplement CONSTRUCTION PLANEX, extending it to handle both concrete and steel-frame buildings (see Chapter 6). The implementations of CONSTRUCTION PLANEX and HARNESS PLANEX have been the most extensive tests of the PLANEX architecture to date.

This implementation of PLANEX includes the KNOWLEDGE SOURCE ACQUISITION MODULE, used to acquire the domain-specific process planning knowledge used by the application systems (see Section 4.4.1). The KNOWLEDGE SOURCE ACQUISITION MODULE is a stand-alone process implemented in KNOWLEDGE CRAFT and COMMON LISP on an EXPLORER. It also includes the REPORT GENERATOR that can format and output a variety of reports, as described in Section 4.4.4. The REPORT GENERATOR is also implemented in COMMON LISP on an EXPLORER.

CONSTRUCTION PLANEX includes GANTT, an interactive scheduling system described in Section 4.4.2. GANTT is implemented in COMMON LISP and KNOWLEDGE CRAFT on an EXPLORER. In addition, CONSTRUCTION PLANEX

includes ANIMATOR, an animation system which is used to illustrate the results of the project planning process (see Section 4.4.5), and INPUT GENERATOR, a modeling system which simplifies the process of inputting the description of the building as required for the planning process. Both of these programs are implemented in C on a SILICON GRAPHICS IRIS Workstation which is linked to the EXPLORER via a local area network. Details of the design and use of these components is included in the CONSTRUCTION PLANEX documentation [116]. CONSTRUCTION PLANEX has also been incorporated into the INTEGRATED BUILDING DESIGN ENVIRONMENT (IBDE) [32], a vertically-integrated set of knowledge-based systems for the design of buildings (see Section 6.3.2).

The architecture underwent another refinement and reimplementation. The major effort in this cycle was to refine the concept of a domain-independent system and to translate PLANEX entirely into COMMON LISP. The resulting implementation has been ported to a number of hardware platforms including the IBM RT PC[®], and IBM PC/AT[®], and a Sun Workstation[®]. CONSTRUCTION PLANEX has been reimplemented using this COMMON LISP version of PLANEX. To maintain device independence, this implementation is based on a *glass-tty* (character-oriented) model of user interaction and thus does not include the graphical interactive scheduling system or the animation subsystem. This version of CONSTRUCTION PLANEX produces activity plans that are used as input to commercial scheduling packages and has been integrated with a PC-based scheduling system. In addition, in the PC environment, a version of the KNOWLEDGE SOURCE ACQUISITION MODULE has been implemented using LOTUS 1-2-3[®].

2 Process Planning and Scheduling

A process plan for construction or manufacturing is the result of a complex cognitive process involving many decisions. Developments in several areas have helped the planner deal with various parts of the process. For example, statistical analysis may be used to estimate activity durations, shortest path algorithms can be helpful in scheduling start times for activities, and bottleneck assignment models may be used to determine which resources should be assigned to specific tasks.

In addition to the advances in mathematical modeling, there have been developments related to the general problem of *planning*. The *process planning problem* involves identifying a set of actions that will achieve a specific goal. Research in the areas of problem-solving and cognitive psychology, automated planning models, and knowledge-based expert systems has contributed to our understanding of the planning problem, but these general methodologies must be augmented with considerable domain-specific knowledge in order to address practical process planning applications.

In this chapter, three approaches to process planning and scheduling are reviewed: (1) classical AI plan formulation systems; (2) optimization models for scheduling; and (3) knowledge-based aids to planning tasks. Although developed in distinctly different fashions, these different approaches are not exclusive but can be complementary and mutually supportive. Moreover, any one approach is unlikely to be sufficient for practical construction and manufacturing process planning. The planning architecture PLANEX incorporates methods and procedures originally developed for these three approaches.

The next two sections focus on the theory of process plan formulation and task scheduling. The third section provides an overview of existing process planning models and methods for construction and manufacturing.

2.1 Plan Formulation with AI Planners

Planning has been an active research area in AI for more than twenty years. Early planning systems focused on the formulation of plans to be executed by robots. Their applications dealt with problems of stacking blocks on a table or moving objects from room to room. Most of these planning systems were written in LISP or in other general-purpose languages. Later, more ambitious systems incorporated developments from other areas of AI. In particular, concepts from expert systems have influenced the development of many recent AI-based planners. Some planning expert systems have been successfully applied in domains such as the generation of plans for genetic experiments and the formulation of plans for manufacturing products [91].

One method of reviewing previous work on AI-based planners is to classify them with respect to the different issues involved in the planning process. An example of such a classification is the very concise categorization of AI planners provided by Tate [102] with respect to these six dimensions:

- *Search space control*—How is the solution space represented and what are the search mechanisms used to obtain a solution?
- *Hierarchy and abstraction levels*—How are problem-solving goals represented at different levels of detail?
- *Goal ordering and interaction detection and correction*—What approaches are used for solving several goals simultaneously?
- *Planning with conditionals and iterators*—How are conditionals or process-constrained relationships within a plan handled?
- *Time and resource handling*—How are time and resource constraints dealt with during the plan formulation process?
- *Domain representation*—What are the appropriate models for the problem-solving operators and constraints?

A deficiency of this classification scheme is that many AI planners, especially the most recent ones, make contributions to several of these categories because these planners use multiple search and representational strategies.

An alternative method for reviewing AI planners is to classify them as *general-purpose* or *domain-dependent* planners. However, this classification seems inappropriate for several reasons. First, although recent planners contain significant knowledge of their application domain, they embody concepts that are applicable to any domain. Second, the applications of *general-purpose* planners such as NOAH [84] required a domain-dependent description of the problem world and operators. Finally, some knowledge-intensive planners incorporate concepts of early *general-purpose* systems like NOAH.

A third alternative is to list AI planners chronologically. However, this type of review is deficient because it does not group the systems according to their common features.

AI PLANNER	TYPE OF PLAN	ARCHITECTURE	CLASSIFICATION (this review)
STRIPS	Linear	Simple	Linear
ABSTRIPS	Linear	Simple	Linear
INTERPLAN	Linear	Simple	Linear
NOAH	Nonlinear	Simple	Nonlinear
NONLIN	Nonlinear	Simple	Nonlinear
DEVISER	Nonlinear	Simple	Nonlinear
MOLGEN	Nonlinear	Layered	Meta Planner
OPM	Nonlinear	Blackboard	Blackboard

Figure 2-1. Classification of Plan Formulation Models

The classification of AI planners used in this review is shown in Figure 2-1. Some of the more interesting AI planning models are classified with respect to two criteria:

- the *type of plans* produced by the model; and
- the overall characteristics of the system's *architecture*.

The first criterion is used to classify planners as *linear* or *nonlinear*. Linear planners produce sequences of ordered actions. Nonlinear planners produce networks of partially ordered actions. While this terminology of *linear* and *nonlinear* plans may be confusing to mathematically inclined readers, it is widely used in the AI planning literature. The second criterion is used to distinguish between different architectures. Some systems are implemented using a primitive or *simple* architecture while other systems use more elaborate structures in which actions are organized into *layers* or into regions of a *blackboard*.

2.1.1 Linear Planners

Planning systems differ in the manner in which they define the solution search space. In some planning systems, the solution space is represented as a network of nodes, where each node represents a complete *state* in the problem world. Nodes are connected by arcs representing actions that transform one state into another. Early AI planners conceived of *planning* as a heuristic search process of this solution space [29, p. 17] which can be summarized as:

Given: An initial situation represented as an object.
 A desired situation represented as an object.
 A set of operators.

Find: A sequence of operators that will transform the initial situation into the desired situation.

Figure 2-2 illustrates this definition of planning. Planning proceeds by searching a state tree for a path that leads from the root node to a leaf node representing the desired state. The final path chosen is the plan or sequence of operators, which when applied to the initial state will achieve the desired goal. Each link in Figure 2-2 represents a specific operator application.

Numerous techniques exist for heuristically searching a state tree. A very simple planning system would explore branches of the tree without knowledge of the manner in which the operators contribute to solving the problem. The difficulties inherent in this kind of a simple planning system are explained by Fikes and Nilsson [33, p. 192]:

In a very simple problem-solving system, we might first apply all of the applicable operators to the initial world model to create a set of successor models. We would continue to apply all applicable operators to these successors and to their descendants (say in a breadth-first fashion) until a model was produced in which the goal formula was a theorem. However, since we envision uses in which the number of operators applicable to any given world model might be quite large, such a simple system would generate an undesirably large tree of world models and this would be impractical.

In effect, a simplistic state tree generation results in an extremely large search tree which is computationally intractable.

During the development of GPS, the *General Problem Solver*, Ernst and Newell [29] developed the general technique called *means-ends* analysis to guide the search. Some of their findings can be summarized as:

- At any point in the planning process, analyzing the *differences* between the desired situation (*ends*) and the current situation is used to select a desirable operator (*means*) from those that are applicable. A desirable operator is one which reduces some of these differences.
- Problem-solving techniques may be embodied in a set of methods that are applied to achieve particular goals.
- Subproblems may be generated by these methods in an attempt to solve the problem.

The development of GPS influenced the further development of AI planners. STRIPS (*Stanford Research Institute Problem Solver*) [33] is capable of generating *linear plans* by reasoning about the differences between the desired and the current states of the world. The search strategy of STRIPS is described by Fikes and Nilsson [33, p. 193] as:

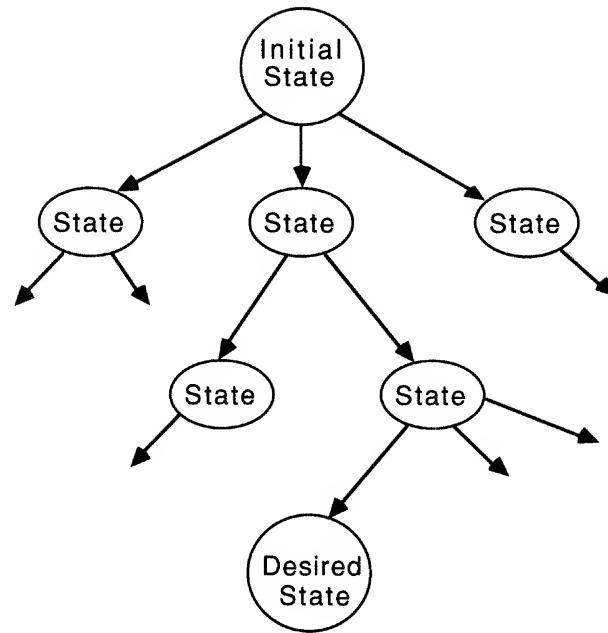


Figure 2-2. Planning by Searching a Tree of States

... we have adopted the GPS strategy of extracting “differences” between the present world model and the goal and of identifying operators that are “relevant” to reducing these differences [...]. Once a relevant operator has been determined, we attempt to solve the subproblem of producing a world model to which it is applicable. If such a model is found, then we apply the relevant operator and reconsider the original goal in the resulting model.

The search of the solution space is performed in a *depth-first* manner with the possibility of *backtracking*. This implies that a particular sequence of operators is pursued until the overall goal or a dead-end is reached. Backtracking is used after reaching a dead-end situation in which no new operators can be found to improve the state. Backtracking reorders the list of goals and explores another branch of the state tree.

The behavior of STRIPS will be illustrated with the three-block problem of Figure 2-3. In this domain, the state of the world is described in terms of two *conditions* or *literals*:

1. *Cleartop X*: This condition is true if there are no blocks on top of block X.
2. *On X Y*: This condition is true if block X is on top of block Y.

The available operators for changing the positions of the blocks are:

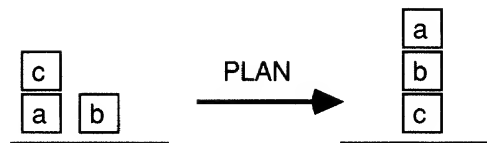


Figure 2-3. Example of a Three-Block Problem

Preconditions	Operator	Effects
(on y x) (cleartop y)	(clear x)	(cleartop x) -(on y x)
(cleartop x) (cleartop y)	(puton x y)	(on x y) -(cleartop y)

Figure 2-4. Description of Operators for the Blocks-World

1. *Clear X*: Take any block from the top of block X and put it on the table.
2. *Puton X Y*: Put block X on top of block Y.

The preconditions and the effects of the operators are shown in Figure 2-4. Thus, the problem of Figure 2-3 becomes:

Given: $\{(on\ c\ a)\ (cleartop\ b)\ (cleartop\ c)\}$

Find: A sequence of operators *clear* and *puton* that will achieve the goals $\{(on\ a\ b)\ (on\ b\ c)\ (cleartop\ a)\}$

STRIPS solves this problem as shown in Figure 2-5:

1. First, it tries to achieve goal $(on\ a\ b)$. The only operator that produces this effect is $(puton\ a\ b)$. In order to apply this operator, the preconditions $(cleartop\ a)$ and $(cleartop\ b)$ must be satisfied.
2. The system introduces the subgoal $(cleartop\ a)$ and tries to achieve this subgoal. The only operator that produces this effect is $(clear\ a)$.
3. The operator is applied, leading to the state labeled "node 1". The subgoal $(cleartop\ a)$ is removed from the list of goals.
4. The system analyzes the initial goals from the new state. It again tries to achieve the goal $(on\ a\ b)$. This may be accomplished by applying the $(puton\ a\ b)$ operator.
5. The operator is applied, resulting in the state labeled "node 2". The system now tries to achieve the subgoal $(on\ b\ c)$. The only applicable operator is $(puton\ b\ c)$. However, in order to apply this operator, its preconditions $(cleartop\ b)$ and $(cleartop\ c)$ must be satisfied.

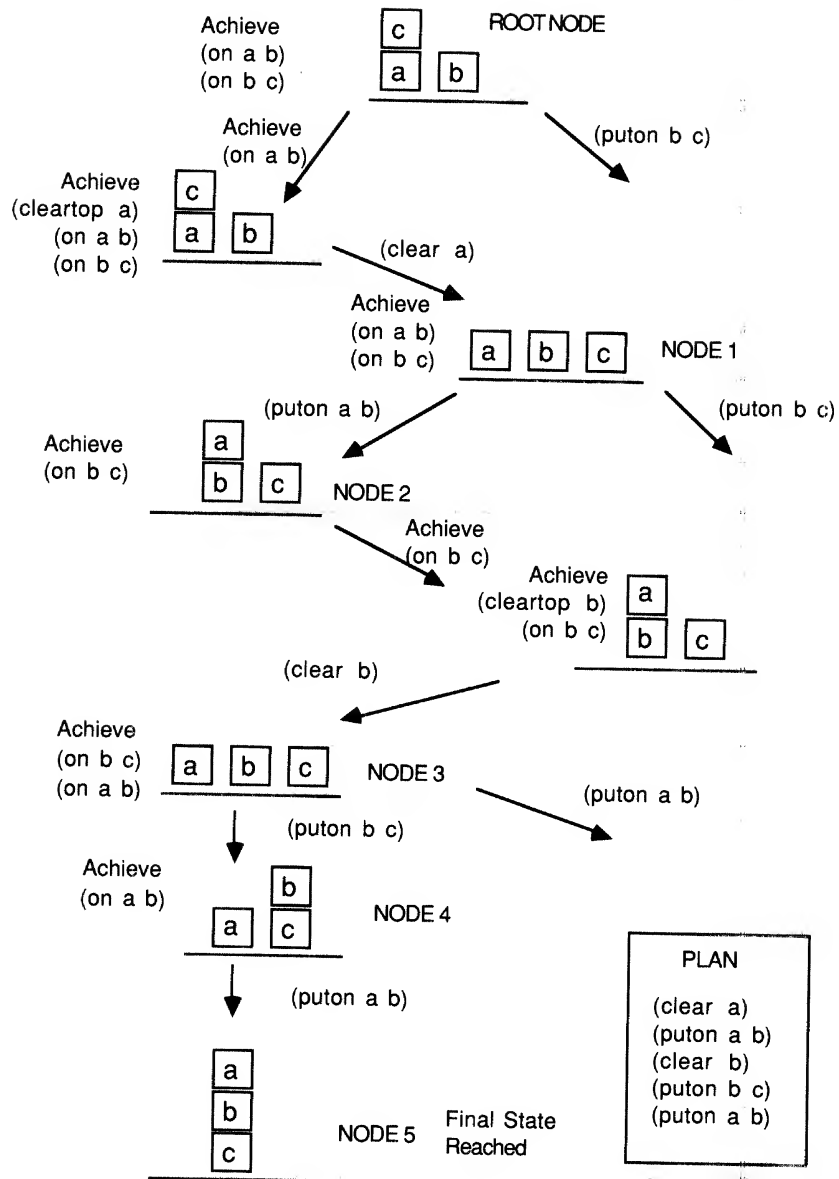


Figure 2-5. STRIPS' Solution to the Three-Block Problem

6. The system introduces the subgoal (*clear top b*) and identifies that the only operator that produces this effect is (*clear b*).
7. When this operator is applied, it produces the state labeled "node 3". The subgoal (*clear top b*) is now removed from the list of goals.
8. Again, the system analyzes the initial goals from the new state. Both final goals are not met. However, this time STRIPS tries to achieve the first goal (*on b c*). The only operator that produces this effect is (*put on b c*).
9. When this operator is applied, it produces the state labeled "node 4". The only unmet goal is (*on a b*). This may be satisfied by applying operator (*put on a b*).
10. Finally, the application of operator (*put on a b*) leads to the desired situation.

STRIPS evolved into ABSTRIPS [84], a planner in which the important concept of planning at different levels of abstraction was introduced. The motivation for planning hierarchically is explained by Sacerdoti [84, p. 412]:

A superior approach to planning would be to search first through an abstraction space, a simplifying representation of the problem space in which unimportant details are ignored. When a solution to the problem in the abstraction space is discovered, all that remains is to account for the details of the linkup between the steps of the solution. This can be regarded as a sequence of subproblems in the original problem space. If they can be solved, a solution to the overall problem will have been achieved. If they cannot be solved, more planning in the abstraction space is required to discover an alternative solution.

In ABSTRIPS, operators from the different levels of abstraction differ only in the number of conditions or literals they possess. An abstract operator has fewer preconditions and effects than a detailed operator. In order to distinguish abstraction levels, literals are ordered using a *criticality* number. In higher levels of abstraction, only the literals with high criticality numbers are considered. This allows the system to produce plans at different levels of abstraction.

ABSTRIPS was able to produce plans more efficiently than STRIPS. However, the system required that the user order the literals by assigning them a criticality number. A different approach to reduce the search effort of STRIPS was developed in INTERPLAN [99, 100]. INTERPLAN analyzes the interactions among the different goals before attempting to achieve any of them. INTERPLAN goals are ordered with respect to the time interval over which they should remain true. Interactions between goals are recorded in a matrix called the *ticklist*. Ticklists provide INTERPLAN with a simple mechanism for backtracking when prior decisions lead to a dead-end.

An example of the use of ticklists in INTERPLAN for the solution of the three-block problem of Figure 2-3 is shown in Figure 2-6 (adapted from [100]). The first steps in INTERPLAN's solution are:

1. It checks whether goal (*on a b*) is satisfied in state 1. The goal is not satisfied; this is indicated with the cross 1 label on the ticklist.
2. The only operator that produces the goal is (*puton a b*), with preconditions (*cleartop a*) and (*cleartop b*). A daughter ticklist is created below the original ticklist.
3. INTERPLAN now checks if subgoal (*cleartop a*) is satisfied in state 1. It is not and the system indicates this with cross 2.
4. The only operator that produces subgoal (*cleartop a*) is (*clear a*). This operator is immediately applied.
5. When operator (*clear a*) is applied, the state of the world changes from state 1 to state 2.
6. In state 2, subgoals (*cleartop a*) and (*cleartop b*) have been satisfied and this situation is indicated with ticks 3 and 4.
7. INTERPLAN applies the operator (*puton a b*) in state 2 and this changes the position of the blocks to that of state 3.
8. The system reexamines the original ticklist and uses tick 5 to indicate that the final goal (*on a b*) is satisfied in state 3 and uses cross 6 to indicate that goal (*on b c*) is not satisfied.
9. The only operator that produces the goal (*on b c*) is (*puton b c*), with preconditions (*cleartop b*) and (*cleartop c*). A daughter ticklist is created below the original ticklist, in which the satisfied goal (*on a b*) is considered *protected*.
10. The system uses tick 7 and cross 8 to indicate the protected goal (*on a b*) and the unsatisfied subgoal (*cleartop b*).
11. The only operator that satisfies subgoal (*cleartop b*) is (*clear b*). This operator is immediately applied.
12. When operator (*clear b*) is applied, the state of the world changes from state 3 to state 4.
13. In state 4, subgoal (*cleartop b*) is satisfied and this is indicated with tick 9. However, the protected goal (*on a b*) is violated. This is indicated with cross 10.
14. The system reexamines the first ticklist.

The remaining steps of the solution are not shown in Figure 2-6. To avoid the protection violation, INTERPLAN reorders the final goals and first satisfies goal (*on b c*) by applying operator (*puton b c*). Finally, it satisfies goal (*on a b*) with the application of operator (*puton a b*). The resultant final plan is: (*clear a*) → (*puton b c*) → (*puton a b*).

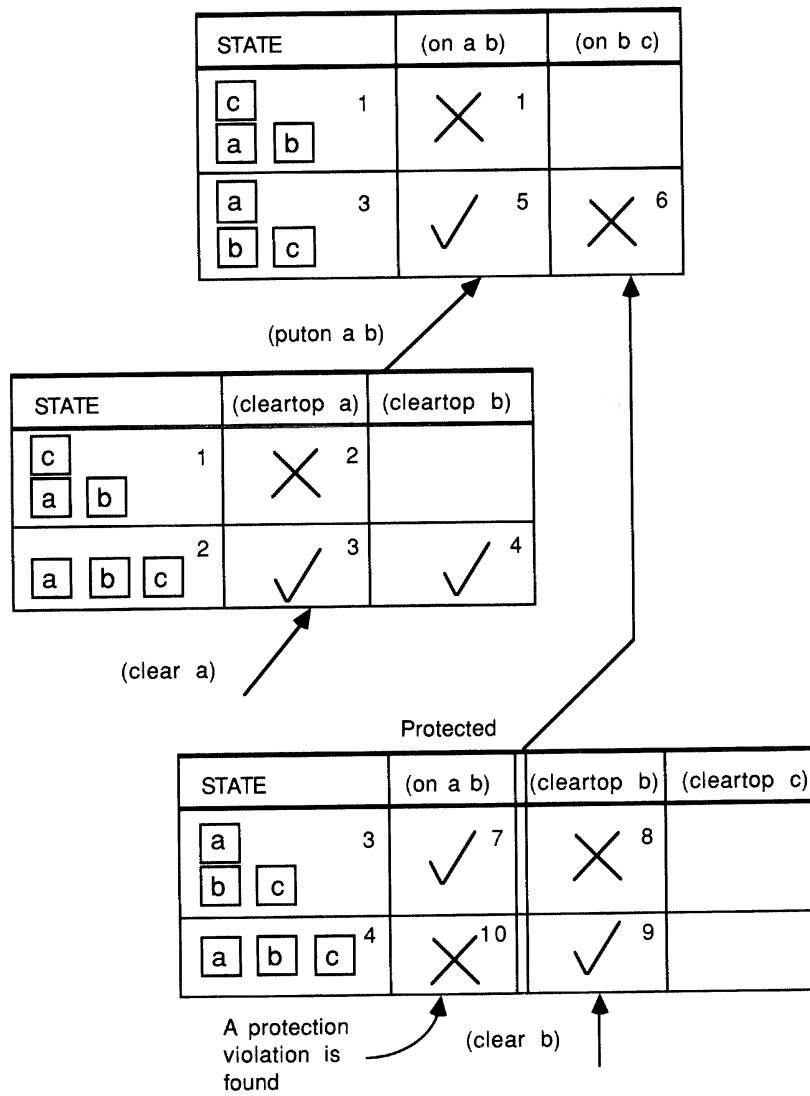


Figure 2-6. INTERPLAN's Ticklists for the Three-Block Problem of Figure 2-3

2.1.2 Nonlinear Planners

The introduction of nonlinear planners in the 1970's marks an important change in the development of AI planning models. The concept of a plan as linear sequences of actions was modified as stated by Sacerdoti [85, p. 206]:

Although the execution of a plan is essentially linear, a plan itself may be thought of as a partial ordering of actions with respect to time.

This is precisely the difference between linear and nonlinear planning, as noted by Chapman [12, p. 2]:

The important idea, due to Sacerdoti, is that a plan (at least while it is being constructed) does not have to specify fully the order of execution of its steps. In other words, a plan is only a partial order of steps; this is what is meant by *nonlinear* planning.

NOAH (*Network Of Action Hierarchies*) [86] uses a model called the *procedural net* in planning. A procedural net is a network of nodes, each of which represents actions at particular levels of abstraction. Similar to previous planning models, each action (node) has an associated list of *preconditions* that must be true to execute the action and a list of *effects* that are added to or deleted from the world model when the action is executed. In addition to these lists, nodes contain a *body* that specifies more detailed actions used to expand the node.

In NOAH, the planning process consists of first creating a small procedural net in terms of abstract actions and then expanding it repeatedly until the plan has been expressed in terms of simple operations or *primitives*. NOAH performs three tasks:

1. *Expand each node of the procedural net.* Each node is expanded into a set of daughter nodes, using the information contained in the body of the parent node. Nodes whose preconditions are already satisfied in the world model are copied as *phantom* nodes, while other nodes are expanded as *goal* nodes.
2. *Apply critics to the new detailed procedural net.* The main use of the critics is to avoid conflicts and redundancies in the new procedural net. An example of a critic is the *resolve conflicts* critic that is used to reorder actions when a precondition of a node is negated by a node in a parallel branch of the procedural net.
3. *Update the world model by adding and deleting objects.* The add and delete lists of the new nodes in the expanded procedural net are used to update the world model.

These three tasks are repeated until the plan has been decomposed into primitive actions and the plan is complete at the most detailed level.

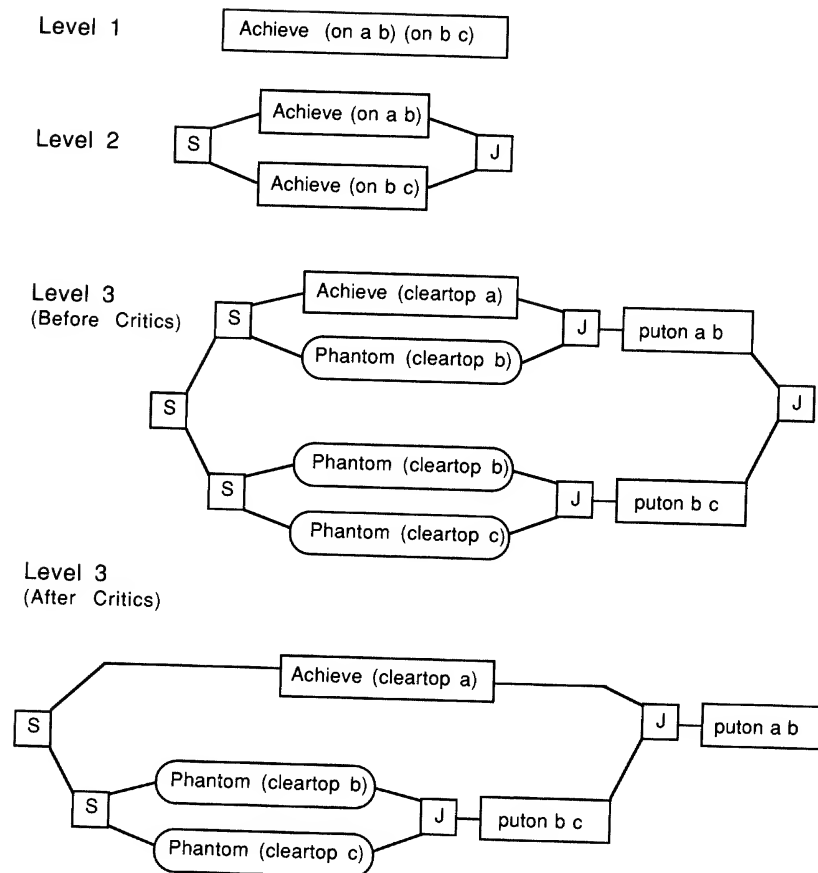


Figure 2-7. NOAH's Solution to the Three-Block Problem of Figure 2-3

An example of the application of NOAH to the three-block problem of Figure 2-3 is shown in Figure 2-7 (adapted from [85]). The first steps of NOAH's solution are:

1. At level 1, the procedural net consists of a single node that indicates the conjunction of goals to be satisfied: *(on a b)* and *(on b c)*.
2. At level 2, NOAH creates a network with two parallel branches with nodes representing the goals to be satisfied. Node *S* and *J* are dummy nodes that indicate a *Split* and *Join* of the network.
3. The system identifies operators (*puton a b*) and (*puton b c*) as possible means to achieve the final goals. Each operator and its preconditions are used to expand the goals of level 2 into the more detailed procedural net of

- level 3. Goal (*on a b*) is expanded into the operator (*puton a b*), the subgoal (*clear a*) and the phantom node (*clear a b*). Goal (*on b c*) is expanded into the operator (*puton b c*), and the phantom nodes (*clear a b*) and (*clear a c*). Conditions (*clear a b*) and (*clear a c*) are represented as phantom nodes because they are true in the initial world model.
4. The system applies critics to the procedural net. NOAH identifies that operator (*puton a b*) negates the phantom node (*clear a b*) in the branch corresponding to operator (*puton b c*). It modifies the topology of the procedural net and establishes a precedence relationship that specifies operator (*puton b c*) precedes operator (*puton a b*). The revised procedural net is shown at the bottom of Figure 2-7.

The last step in NOAH's solution is not illustrated in Figure 2-7. NOAH proceeds to expand the unsatisfied goal (*clear a*) and identifies the operator (*clear a*) with preconditions (*on c a*) and (*clear a c*) as a means to achieve the goal. Then the system recognizes that operator (*puton b c*) negates the precondition (*clear a c*) and it establishes a precedence relationship that specifies operator (*clear a*) precedes operator (*puton b c*). The final plan is a network with three precedences: (*clear a*) \rightarrow (*puton b c*), (*puton b c*) \rightarrow (*puton a b*) and (*clear a*) \rightarrow (*puton a b*).

NONLIN [101] is another *NONLINEar* planner that was created as an evolution of INTERPLAN. In NONLIN, each action is described by a *task schema* using a *Task Formalism*. Task schemas contain information about the effects of an action, the preconditions that have to hold before the action is performed, and the manner in which the action is expanded into lower-level actions. Conditions are classified into the following categories:

- *Unsupervised Conditions*, which must exist before a task is finished but are the responsibility of other actions;
- *Supervised Conditions*, which must exist before a task is finished and are the responsibility of the action being considered; and
- *Use-When Conditions*, which are static in the sense that they do not depend on any action. However, they must hold before an action is executed.

This classification of conditions enriches the declarative representation of tasks and improves the problem-solving behavior of the planner. NONLIN uses the ticklists of INTERPLAN more effectively and is capable of solving problems which NOAH cannot solve.

NONLIN has been applied in several different domains. One application of particular interest is the formulation of construction project networks. NONLIN expands aggregate project networks into detailed networks using knowledge stored in task schemas. The input to the system is an aggregated action such as *erect structure* that NONLIN expands repeatedly. The output is a network of activities that do not require further expansion. NONLIN does not distinguish

```

ACTSCHEMA DECOR
  PATTERN <<DECORATE>>
  EXPANSION
    1 ACTION <<FASTEN PLASTER AND PLASTER BOARD>>
    2 ACTION <<POUR BASEMENT FLOOR>>
    3 ACTION <<LAY FINISHED FLOORING>>
    4 ACTION <<FINISH CARPENTRY>>
    5 ACTION <<SAND AND VARNISH FLOORS>>
    6 ACTION <<PAINT>>
  ORDERINGS 1 ---->3 6 ---->5 SEQUENCE 2 TO 5
  CONDITIONS
    UNSUPERVISED <<ROUGH PLUMBING INSTALLED>> AT 1
    UNSUPERVISED <<ROUGH WIRING INSTALLED>> AT 1
    UNSUPERVISED <<AIR CONDITIONING INSTALLED>> AT 1
    UNSUPERVISED <<DRAINS INSTALLED>> AT 2
    UNSUPERVISED <<PLUMBING FINISHED>> AT 6
    SUPERVISED <<PLASTERING FINISHED>> AT 3 FROM 1
    SUPERVISED <<BASEMENT FLOOR LAYED>> AT 3 FROM 2
    SUPERVISED <<FLOORING FINISHED>> AT 4 FROM 3
    SUPERVISED <<CARPENTRY FINISHED>> AT 5 FROM 4
    SUPERVISED <<PAINTED>> AT 5 FROM 6
END;

```

Figure 2-8. NONLIN's Task Formalism for a Construction Activity

between a construction activity and a component of the design. Figure 2-8 (adapted from [101]) shows an example of a task schema representing the construction activity <<decorate>>. The schema contains knowledge describing how the aggregate activity can be expanded into the network of subactivities shown in Figure 2-9. This network is described in the *expansion* and *orderings* fields of the task schema. The information in the *conditions* field indicates that five unsupervised conditions must exist before the task is finished, and that five supervised conditions are created when the activity is expanded. Unsupervised conditions are not asserted by the <<decorate>> activity. However, they must be satisfied before executing some of the subactivities. For example, the <<drains installed>> condition has to be satisfied before the <<pour basement floor>> subactivity is executed. In contrast, supervised conditions are asserted within the expansion of the activity. For example, the <<painting>> condition is asserted by the <<paint>> action, and it must be true before the <<sand and varnish floor>> action is executed.

Another nonlinear planner which followed the initial development of NOAH and NONLIN is DEVISER, a system developed by Vere [104] for use in the planning and scheduling of an autonomous unmanned spacecraft. DEVISER was the first system to consider time constraints in the generation of plans. More recent versions of NONLIN have also incorporated time and resource constraints during the planning process. In DEVISER, time constraints are used

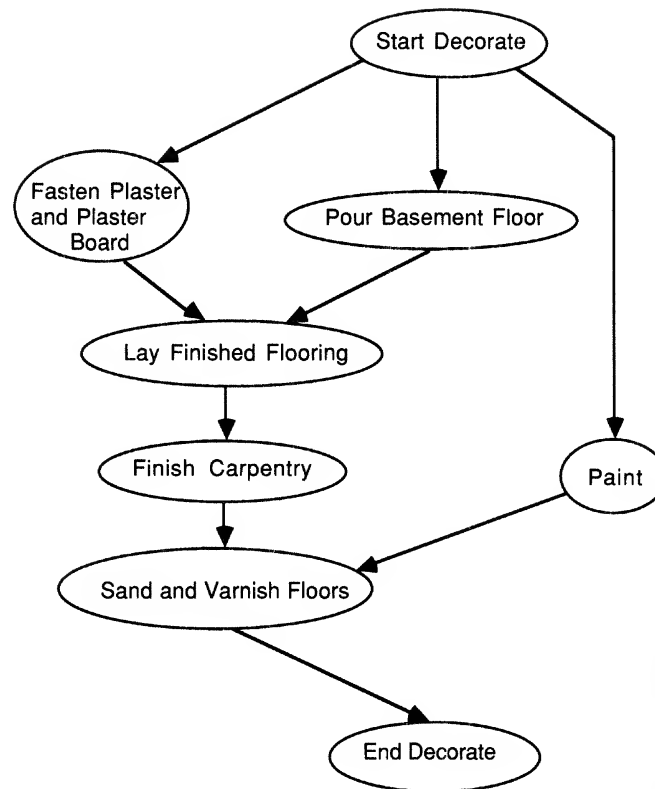


Figure 2-9. NONLIN's Subnetwork for a Construction Activity

to specify when sets of goals should be satisfied and how long the goal conditions should be preserved. The output of DEVISER is a network with bounds for the start times of the activities. To represent these constraints, nodes of the network are divided into two major categories:

- *Activities* whose realization in time is determined during the planning process. Activities are either: (1) *Actions* that have to be explicitly activated and incorporated into the planning network; (2) *Events* that are triggered spontaneously by changes in the state of the world; or (3) *Inferences* that are propositions whose truth changes depending on the value of the variables considered.
- *Scheduled Events* whose occurrence is fixed in time regardless of the structure of the plan. Scheduled events can be considered additional constraints on the planning process.

Representation of activities in DEVISER is similar to that in NOAH. Each activity is described by specifying: (1) *Type* (action, event or inference);

(2) *Context*, a set of *literals* representing the preconditions that must be true before the activity can be executed and that are not altered by the activity; (3) *Antecedent*, a set of literals representing conditions that will be deleted from the world model when the activity is executed; and (4) *Consequent*, a set of literals that will be added to the world model when the activity is executed. Antecedent and consequent lists resemble the add and delete list of NOAH. Activity durations may also be specified; these may be a constant or computed by evaluating a function.

An important feature of DEVISER is that it uses different types of literals to describe the world. Literals are either:

- *Ordinary*, whose truth is determined by accessing a distributed data store. Their value may be changed by the antecedent or consequent part of an activity.
- *Procedurally Defined*, whose truth is determined by calling a procedure. They cannot appear in the antecedent or consequent list of any activity.
- *Functionally Determined*, specified as a list in which the rightmost term is a function of all the other terms.

The planning process in DEVISER is very similar to that used in NOAH to expand procedural nets. Planning consists of repeating three tasks:

1. *Linking*. When the assertions of a node *J* are satisfied by another node *I*, a link from node *I* to node *J* is added to the network, and node *J* becomes a phantom node. This is similar to NOAH's definition of phantom nodes.
2. *Node expansion*. When a goal or action cannot be achieved, it is decomposed into a set of daughter nodes. The goal's preconditions become goals and side effects are added to the assertions of the expanded node.
3. *Conflict detection and resolution*. Conflicts between activities in parallel branches are identified and resolved by reordering the activities. Conflicts must be resolved each time the network is expanded.

During the planning process, DEVISER ensures that all conditions (context and antecedent) of each activity are satisfied throughout the duration of the activity. However, the following two cases are of interest: (1) When one of the antecedent conditions is deleted immediately after starting the activity; and (2) When *trigger* preconditions are present (they need to be present only when an activity is started). To handle these cases, DEVISER decomposes activities into two nodes, one representing the start of the activity with a zero duration, and another consecutive node representing the execution of the activity.

Activity start and end times are calculated by assigning a *window* to each activity. A window is a list with three time values for the activity:

<u>Type</u>	<u>Constraint</u>	<u>Description</u>	<u>Window</u>	<u>Lag</u>
1	At t	Instantaneous event	(t,t,t)	0
2	Before t	Starting in (0,t)	(0,nil,t)	t
3	After t	Starting in (t,infinity)	(t,nil,infinity)	infinity
4	Between t1,t2	Starting in (t1,t2)	(t1,nil,t2)	t2-t1
5	Between t1,t2 best t3	Starting in (t1,t2) and preferably at t3	(t1,ideal,t2)	t2-t1

Figure 2-10. Time Constraints in DEVISER

1. *Earliest-Start-Time* (EST);
2. *Ideal-Start-Time* (IST); and
3. *Latest-Start-Time* (LST).

Lags are computed by subtracting EST from LST. At the beginning of the planning process all activities have a window (0,nil,infinity) representing the constraint that the start time is positive. The user may specify additional windows for goals and scheduled events. Time constraints that may be specified using windows are illustrated in Figure 2-10. As the planning process proceeds, activity windows and subgoal windows are computed dynamically. Changes in window duration occur when linking activities, expanding nodes or reordering activities. Window duration changes are propagated recursively to neighboring nodes until the effects of a duration change have been fully propagated throughout the network.

DEVISER uses windows to classify preconditions into *packages* based on their time constraints. The set of conditions in a package must be satisfied simultaneously by all preconditions included in the package, and must satisfy all the time constraints (window specifications). Windows are also used to resolve conflicts and to order goals. The ideal time of an activity is never modified; it is used to order the activities should there be a "tie" between alternatives.

DEVISER generates feasible plans. It is capable of backtracking when the earliest-start-time of an activity is greater than its latest-start-time. However, it does not contain any mechanism for relaxing constraints if all constraints cannot be satisfied.

2.1.3 Meta-Planners

Meta-planners represent a different approach in the development of AI-based planning models. These systems are concerned with the order in which the different *planning operations* are executed, rather than with the successive expansion of *plan actions*. The distinction between planning operations and plan actions was noted by Stefik [95, p. 141]:

The decision-making knowledge is organized in a layered control structure which separates decisions about the planning problem from decisions about the planning process. The approach, termed meta-planning, exposes and organizes a variety of decisions, which are usually made implicitly and sub-optimally in the planning programs with rigid control structures.

In meta-planners, planning operators and data structures are divided into *layers*. Operators of a given layer control the execution of operators of the layer immediately below it. In addition, operators of the upper layers are distinct and more general than operators in the lower levels. Communication between the layers is usually through a message interface.

MOLGEN (*MO*Lecular *GE*Nerator) [94, 95] is a meta-planning system developed by Stefik to plan genetic experiments. The system generates plans by dividing the planning problem into subproblems and analyzing the interactions among these subproblems. This behavior is inspired by the means-ends analysis technique of GPS. However, MOLGEN incorporates *constraint* objects to explicitly represent interactions among subproblems. MOLGEN handles constraints through a *constraint posting cycle* in which the following tasks are executed:

- *Constraint Formulation* adds constraints to a plan as it is detailed;
- *Constraint Propagation* creates new constraints from old constraints in the plan; and
- *Constraint Satisfaction* finds values for a group of variables that satisfy a given set of constraints.

These tasks are continuously repeated throughout the planning process.

MOLGEN's planning architecture divides planning operators and planning objects into the four layers shown in Figure 2-11 (adapted from [93]). The system uses this structure to control the execution of the constraint posting cycle. The layers are:

1. *Laboratory Layer*. This is the lowermost layer. The operators in this layer do not control any other operators. Laboratory operators are activated by messages from the design operators. Laboratory operators model modifications to physical objects such as a gene or a bacterium. Within this layer, operators are structured in a two-level hierarchy of abstract and specific operators. Four abstract operators exist: (1) *Merge* combines objects; (2) *Amplify* increases the amount of something; (3) *React* changes properties; and (4) *Sort* separates something into its components. These operators, called the MARS operators, are further refined into specific operators. For example, the abstract *Merge* operator may be refined into the specific operators *Transform* and *Screen*.

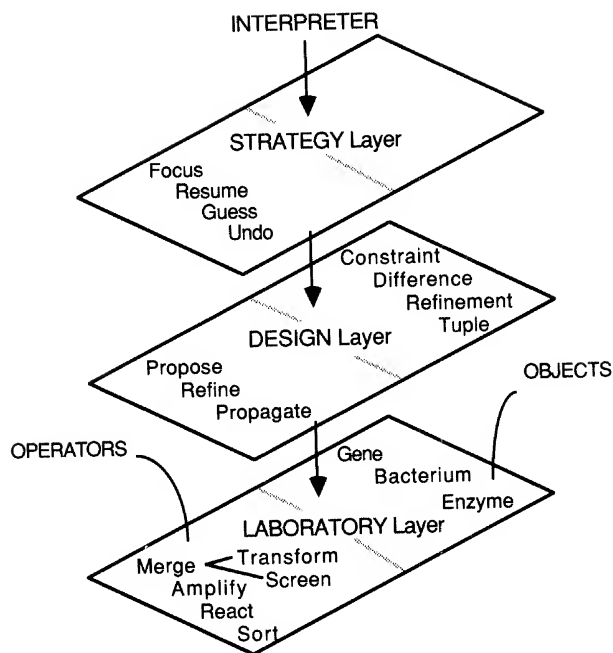


Figure 2-11. MOLGEN's Layered Structure

2. *Design Layer*. The operators in this layer control the execution of the operators in the laboratory layer. Objects in this layer do not model physical objects, but represent constraints, differences, refinements and tuples. There are three types of operators in the design layer:
 - a. *Comparison Operators* are used to evaluate partial solutions. First, laboratory goals are examined by comparing objects with their prototypes in order to find unusual features. Then predictions are compared with goals to check for mismatches and backtrack if necessary.
 - b. *Temporal-Extension Operators* are used to propose extensions to the plan given the actual differences between the state of the system and the goals. MOLGEN: (1) proposes operators to reduce the difference by asking the MARS operators which of them is feasible; (2) proposes goals for these operators; and (3) predicts the results by requesting a one-step simulation in the laboratory layer.
 - c. *Specialization Operators* are used to expand operators into more detailed ones and to propagate constraints backwards in time.
3. *Strategic Layer*. The strategic layer creates and executes design tasks in a manner similar to the way the design layer controls the laboratory operators.

Only four general strategic operators are used: (1) *Focus* simulates a design task; (2) *Resume* continues suspended tasks; (3) *Guess* uses heuristics to select a design task; and (4) *Undo* backtracks when necessary. These operators are activated by messages from the *strategic-design* interface. At any point in the planning process, design tasks are either *done*, *failed*, *suspended* or *canceled*.

4. *Interpreter*. This is the uppermost layer. The interpreter executes strategic operators.

2.1.4 Blackboard Planners

Blackboard planners constitute yet a different type of AI planning system. They plan by using mechanisms similar to those of the *blackboard* problem-solving framework developed in HEARSAY-II [28]. An example of such a blackboard planner is OPM [45], a system developed by Barbara Hayes-Roth for multi-task planning problems. Multi-task planning involves selecting and ordering several plan tasks from a list of desired tasks. This problem is a special case of the *general control problem* [45, p. 251]:

The control problem—which of its potential actions should an AI system perform at each point in the problem-solving process?—is fundamental to all cognitive processes.

The architecture of OPM is shown in Figure 2-12. The blackboard is a global data structure where all solution elements generated during the problem-solving process are stored. These solution elements are generated by a set of independent and cooperative processes called *knowledge sources*. Knowledge sources are independent because they do not directly invoke any other knowledge source. They may only influence the behavior of other knowledge sources by altering the information stored on the blackboard. In addition, knowledge sources are cooperative because they contribute to solve the same problem. Knowledge sources have three basic components:

- *Trigger Conditions* used to identify those knowledge sources that may be helpful at a particular point in the planning process;
- *Preconditions* used to impose restrictions on the ability to invoke one of the knowledge sources; and
- *Actions* used to specify the changes that the knowledge sources make on the blackboard.

When the trigger conditions of a knowledge source are satisfied by events on the blackboard, a *Knowledge Source Activation Record (KSAR)* is created and it is introduced into a set of pending KSARs called the *To-Do-Set*. A KSAR is a unique combination of a knowledge source and the objects from the blackboard

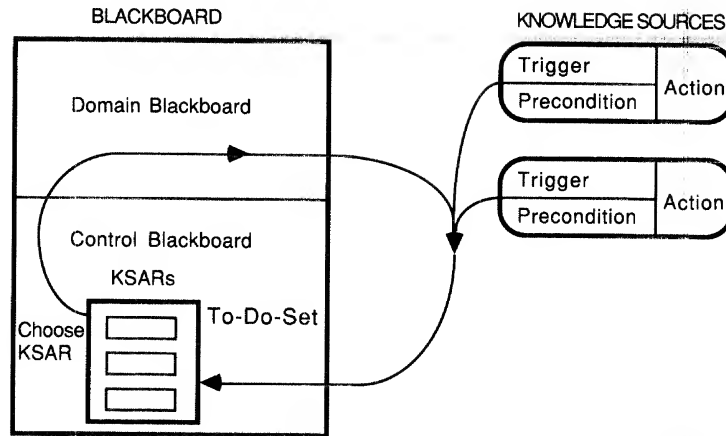


Figure 2-12. OPM's Blackboard Architecture

used to evaluate the trigger conditions. In each problem-solving cycle, a KSAR of the *To-Do-Set* is selected for execution using a special knowledge source entitled *Choose-KSAR*. When a KSAR is executed, the actions of its associated knowledge source are applied to the associated objects in the context. Different KSARs for the same knowledge source represent applications of the same actions to different objects on the blackboard.

OPM employs two independent blackboards:

- the *domain* blackboard, which contains solution elements related to the problem, such as plan tasks and task precedences; and
- the *control* blackboard, which contains solution elements related to the planning process, such as problem-solving goals and pending KSARs.

Each blackboard is organized with respect to plan execution time intervals and different levels of plan abstraction. The domain blackboard is divided into four levels of abstraction: (1) *outcome* contains information about which activities are included in or excluded from the plan; (2) *design* stores temporal specifications for the whole plan; (3) *procedure* contains sequences of individual tasks; and (4) *operation* contains completely specified activities.

The control blackboard is divided into six levels of abstraction: (1) *problem*, with the definition of the global problem goals to be achieved; (2) *strategy*, with information about the general sequential plan for solving the problem; (3) *focus*, with definition of local problem goals; (4) *policy*, with information about the global scheduling criteria; (5) *To-Do-Set*, containing a list of pending KSARs; (6) *chosen-action*, specifying the KSAR chosen to be executed.

Knowledge sources are also divided into: (1) *domain* knowledge sources; and (2) *control* knowledge sources. Examples of domain knowledge sources are

those responsible for refining an abstract plan into a more detailed one. Examples of control knowledge sources are those responsible for choosing which pending KSAR to execute.

2.1.5 The Frame Problem

The previous discussion outlined the characteristics of a number of classical AI planning systems. These systems are capable of identifying plans to achieve predefined goals in highly restricted planning domains. Before introducing the role of optimization methods for planning, it is worthwhile to note a general problem that limits the capability of any automated planning, namely how to identify the problem state changes as operators are applied. This general identification problem has been called the *frame problem* by Brown [9] and McCarthy and Hayes [68]. The problem is intimately connected to the practicality of making truthful inferences as well as effective plans. When an operator has been applied or an inference has been made, the new description of the world must reflect the direct changes caused by the operation as well as the changes in all the facts derived from the altered situation.

As a simple example of the frame problem, consider a block containing a hole as shown in Figure 2-13 (a). If a horizontal cut is made below the hole, the hole is removed (Figure 2-13 (b)). The description of the block should now include the fact that both a portion of the block and the hole have been removed. However, a different cut would not remove the hole (as in Figure 2-13 (c)). Inferring whether or not the hole exists after a cut would be difficult in an automated planner. For example, operators in planners such as STRIPS or NOAH are defined by the required preconditions and post-operation facts. With this simple scheme for defining operations, determining whether or not a hole disappears is laborious.

As operators are applied, one mechanism for recognizing effects is to record conditional dependencies. This approach has been generalized as a *truth maintenance system* used to record and maintain all interdependencies among conditions [24, 25, 34]. For example, the existence of the hole in the block (Figure 2-13 (a)) depends upon the existence of the block itself. If a portion of the block disappears the precondition for the hole's existence must be reexamined.

The explicit representation of dependencies among conditions permits *dependency-directed backtracking* [92]. As a particular condition is altered, the conditions or facts derived can be identified. Subsequent operators and searches can focus on these alterations. However, identifying the dependencies among conditions typically requires considerable knowledge. For example, determining whether any or all of the hole remains in a block after a cut (Figure 2-13) requires considerable geometric analysis.

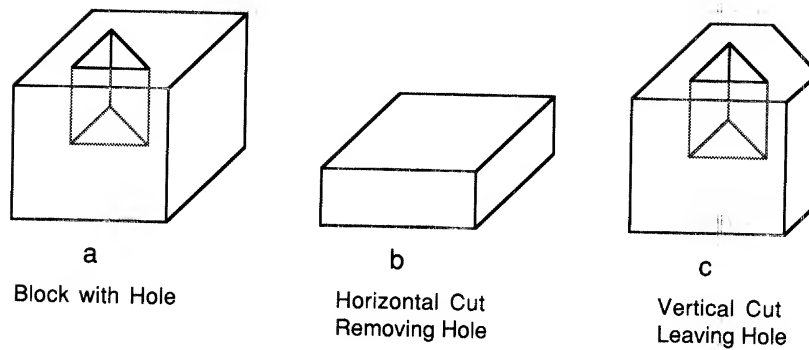


Figure 2-13. A Block with a Hole and Cuts

Coping with the frame problem is a major concern in the continuing development of classical AI planners. For example, the planning system SIPE (*System for Interactive Planning and Execution Monitoring*) [112] includes: (1) "resource reasoning" to insure that operators using common resources are ordered correctly; (2) a "truth criterion" which employs MOLGEN-type constraints (see p. 30) and domain knowledge to avoid at least in part the frame problem; and (3) explicit and (presumably) comprehensive representation of operator preconditions and effects as in STRIPS (see p. 17). Nevertheless, these heuristic methods cannot guarantee that the resulting plans will overcome the frame problem and produce workable plans. Coupled with the need for extensive domain knowledge, this observation motivates including extensive user interaction and review mechanisms in PLANEX.

2.2 Project Scheduling with Optimization Methods

In planning the execution of a project, it is important to identify both the activities to be executed and the precedence restrictions among these activities. However, formulating a network of activities is only one of the tasks performed by planners. There are other questions that must be answered before a project is executed. For example:

- *what* type of resources (labor, equipment and materials) should be assigned to the activities?
- *how much* of these resources should be used by each activity?
- *when* should each activity start?

Usually there are many possible answers to these questions. During the scheduling process, the planner has to select a set of answers that seems appropriate. After this process is completed, the set of decisions made by the planner are reflected in a *project schedule* that is used to monitor and control the execution

of the project. Fortunately, there are numerous algorithmic procedures to aid in scheduling. The capabilities of these procedures motivates their inclusion in the PLANEX architecture.

The computation of project schedules has evolved from the early scheduling models such as the *Critical Path Method* (CPM) and the *Project Evaluation and Review Technique* (PERT) into more sophisticated models that incorporate additional aspects of the scheduling problem [50, 111]. Project managers have recognized the advantages of using some of these models to assist them with the scheduling process. Today there are several commercial packages that assist planners in preparing and maintaining project schedules [2, p. 231]:

Microcomputer-based project management software systems have evolved from simple packages focusing on basic capabilities, such as Gantt-charts, to sophisticated systems offering resource management, progress reporting, and superior input and output interfaces. The commercially available packages are becoming increasingly easier to use and more appealing visually.

Models for project scheduling can be divided into two categories:

- *deterministic* models which assume activity durations are fixed or are expressed as a function of the cost incurred in completing the activities; and
- *probabilistic* models which assume activity durations are stochastic variables with particular probability distributions.

This distinction is important for several reasons. The information obtained from deterministic models is different from that provided by probabilistic models. Deterministic models are usually applied to *compute* schedules that minimize total completion time or total budget. Probabilistic models are used to *estimate* the total completion time of a project or to *simulate* the execution of a project. For the most part these models have evolved independently [111]. There are some models that incorporate uncertainties into the CPM [13, 103]. However, most deterministic scheduling models are descendants of the original CPM model [57], while probabilistic models are extensions of the PERT model [66].

The remainder of this section provides a brief review of some relevant project scheduling models. The review focuses on *deterministic* scheduling models. More extensive reviews may be found in other books [26, 72, 110] and papers [41, 111].

2.2.1 Classification of Deterministic Scheduling Models

Deterministic scheduling models are used to sequence activities in time to achieve managerial goals. In most deterministic scheduling models, the goal is to minimize the total completion time of the project. However, some models

introduce other goals related to the variability in resource requirements or the total budget allocated to the project. Models differ not only by the intended goals, but also by the type of *constraints* considered and by the *assumptions* incorporated. For example, some models do not consider limits on the number of resources consumed while others impose strict restrictions on the allocation of resources. Similarly, some deterministic scheduling models assume that activity durations are fixed while others assume that the duration of an activity decreases as the cost of the resources used to perform the activity is increased.

This work classifies deterministic scheduling models according to the way they consider the following parts of a project schedule: (1) network topology; (2) activity durations; (3) activity costs; and (4) resource consumption profiles. Models are classified into the following categories, as shown in Figure 2-14:

- *Models with fixed activity durations.* These models assume that activity durations are known. Resources are assumed to be unlimited and the network topology is fixed.
- *Models for time-cost trade-offs.* In these models, activity durations are expressed as a function of activity costs. There may be constraints on the total completion time of the project or on the total budget. However, these models do not impose limits on the resource consumption profiles. These models do not permit changes to the network topology.
- *Models with resource considerations.* These models are used to schedule projects when resources are limited or when peak resource requirements must be reduced. These models do not permit changes to the network topology.
- *Models for combined planning and scheduling.* These models combine planning and scheduling in a unified framework. The major distinction from the other models is that the topology of the network may be changed.

Deterministic Scheduling Models

	Fixed Duration	Time-Cost Trade-Offs	Resource Considerations	Planning & Scheduling
Network Topology	Fixed	Fixed	Fixed	Variable
Activity Durations	Fixed	Duration and Cost related	May vary	May vary
Activity Costs	Not considered	Duration and Cost related	May vary	May vary
Resource Profiles	Assumed Unlimited	Assumed Unlimited	Upper limits may exist	Upper limits may exist

Figure 2-14. Classification of Deterministic Scheduling Models

The four types of models are not independent. Fixed duration models may be considered to be a special case of time-cost trade-off models. Similarly, both of these models are specializations of resource constrained models for the case of arbitrarily large resource profiles. However, this work treats the four types of models separately because the nature of the mathematical programs used in the models is different. Fixed duration models can be solved with simple shortest path algorithms⁴, while resource constrained models involve more elaborate algorithms to solve integer programming problems.

2.2.2 Models with Fixed Activity Durations

Although the original CPM model allowed variable activity durations [57], common applications use a simplified version of the model in which activity durations are considered fixed. This model, called the *Basic CPM* model (BCPM), is used to find the earliest project completion time, the *slacks* or *floats* associated with the project activities, and the set of *critical* activities.

Common implementations of the BCPM model use an *Activity-On-Branch* (AOB) diagram to represent the project. In this representation, branches⁵ represent activities and nodes represent events. The earliest completion time of the project is found by computing the longest path from the node representing the start of the project to the node representing project completion. Figure 2-15 (adapted from [110, p. 6]) shows an AOB diagram for a project composed of five activities. The numbers above the arrows represent the durations of the activities (in days). The earliest completion time of the project is 35 days.

Several operations research (OR) algorithms can be used to compute the longest path in a network [61, p. 3, 14, p. 8]. In acyclic networks, a popular algorithm, developed by Dijkstra [61, p. 71], provides the basis for the *forward* and *backward* pass computations used to solve the BCPM [72, p. 74]. However, there are other OR techniques that can be used for this purpose. In particular, the model may be solved using *linear programming* (LP) or *network flow* algorithms. Network flow formulations are also useful in solving time-cost trade-off models and resource-constrained models. These applications are described below.

Another deterministic scheduling model that assumes fixed activity durations is the *Precedence Diagramming Method* (PDM) [72]. The basis for this model was a report by Fondahl [35] in which *Activity-On-Node* (AON) diagrams were used to represent project networks. In these diagrams, nodes represent activities

⁴ Scheduling requires that the *longest* path be computed. This involves simple modifications to shortest path algorithms.

⁵ In this discussion, the terms *branch*, *link* and *arc* are equivalent.

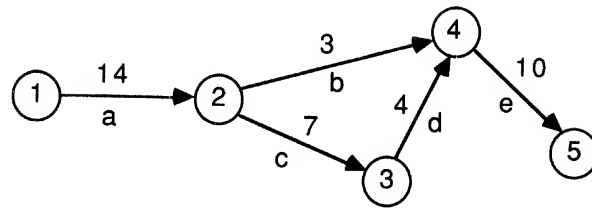


Figure 2-15. Example Project Network (AOB diagram)

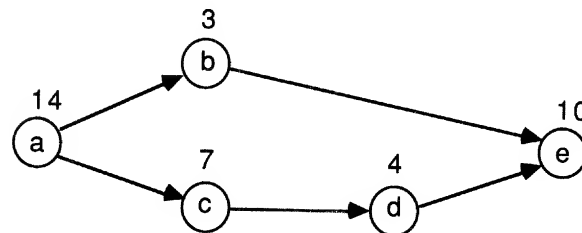


Figure 2-16. Example Project Network (AON diagram)

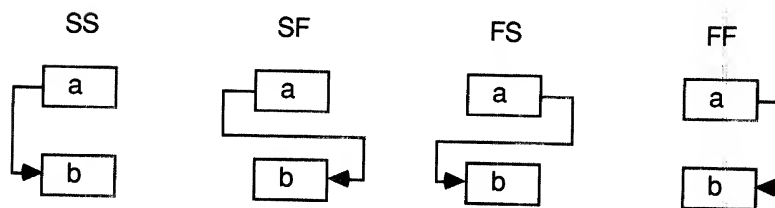


Figure 2-17. Different Types of Precedences in the PDM

and arcs represent precedence relationships. The BCPM is easily adapted for an AON representation. Given an AON network representation, the BCPM finds the longest path between *start* and *finish* activities.

Figure 2-16 shows the AON diagram for the example project of Figure 2-15. In this diagram, activity durations are indicated by the numbers above the nodes and the critical path includes the nodes *a*, *c*, *d* and *e*. Arrows indicate *Finish-to-Start* (FS) precedences: the successor activity cannot *start* until the predecessor activity has *finished*. The PDM incorporates other precedences in addition to the FS relationships of the BCPM. There may be *Start-to-Finish* (SF), *Finish-to-Finish* (FF) and *Start-to-Start* (SS) relationships between the network activities, as shown in Figure 2-17.

There are several deficiencies associated with the PDM. First, the interpretation of floats is not as straightforward as in the BCPM model. Second, there may be anomalies such as the delay in the completion time of the project

due to the reduction of an activity duration [110, p. 145]. Finally, activity *splitting*, which may improve the schedule, is not a feature of the original PDM formulation. In order to solve these problems, some algorithms that include automatic splitting have been proposed [73]. Nevertheless, the PDM facilitates the introduction of additional types of precedence relationships between activities (e.g., start-to-start precedences) and obviates the need for dummy links to maintain proper network topologies [50].

2.2.3 Models for Time-Cost Trade-Offs

The objective of the original CPM model was described as [57, p. 297]:

The mathematical model upon which the Critical-Path Method is based is a parametric linear program that has the objective of computing the utility of a project as a function of its duration. For each feasible project duration, a feasible project schedule is obtained that has maximum utility among all feasible schedules of the same project duration.

To solve the model using an LP formulation, it is assumed that the time to perform an activity decreases linearly as the cost of performing the activity increases. Figure 2-18 shows this linear relationship between activity duration and associated cost. The *normal* time is the time required to perform the activity if no additional resources (e.g., overtime hours or additional labor) are invested. The *crash* time is the minimum time required to complete the activity. This represents the situation in which all possible additional resources have been invested, and the activity is completed in the least possible time.

When the time-cost relationship is linear, a LP formulation can be used to solve for time-cost trade-offs. Parametric LP may be used to obtain a plot of the overall project cost versus the project completion time. This plot, called the *project cost curve*, may also be obtained by formulating the problem as a network-flow problem in which each activity is represented by two parallel arcs [36]. When time-cost relations are linear, the project cost curve is piecewise convex.

Obviously, time-cost relations need not be linear. For cases in which they are convex, OR techniques such as the Frank-Wolfe method can be used to solve the problem [47]. The CPM with concave time-cost relations is more difficult and was studied by Falk and Horowitz [30]. Their algorithm is based on a *branch-and-bound* approach in which successive LPs are solved. This algorithm can be used for problems in which piecewise convex time-cost relations are combined with piecewise concave functions. For the general case in which the time-cost function is nonmonotonic, Elmaghraby [26, p. 108] describes an algorithm based on *dynamic programming* (DP) techniques.

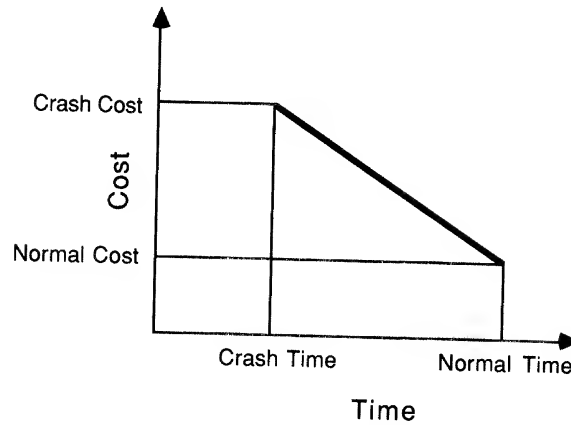


Figure 2-18. Linear Time-Cost Relationships for an Activity

An additional aspect of the time-cost trade-off problem arises with the introduction of constraints on event completion times. Elmaghraby and Pulat [27] developed an efficient algorithm to compute optimal project duration compression with linear time-cost relations and linear penalties for tardiness.

2.2.4 Models with Resource Considerations

Several scheduling models that consider the resource requirements of project activities have been studied [23]. There are two principal types of models for this problem:

- *resource leveling models* that smooth resource consumption profiles; and
- *resource allocation models* that schedule activities when there are constraints on the total amount of available resources.

Both types of models usually involve more complex mathematical formulations than the simpler scheduling models that assume fixed activity durations or that have known time-cost trade-offs. The approaches used to solve project scheduling problems with resource considerations may be classified into:

- *optimization* methods that search for the best schedule; and
- *heuristic* methods that search for a good schedule.

Optimization methods have been applied only to small projects because of the combinatorial nature of the problem. Heuristic methods have been successfully applied to real problems and are included in sophisticated scheduling packages [2].

One heuristic resource leveling algorithms, proposed by Burgess and Kilbreth [10], uses the sum of the squares of daily resource requirements as a

measure of effectiveness of the leveling process. Activities are scheduled sequentially, starting from the last activity and proceeding to the first, in order to reduce this variance. A different approach was proposed by Levy, Thompson and Wiest [64]. In their approach, daily resource requirements are computed on the basis of an earliest start schedule. The algorithm tries to reduce peak resource requirements by setting *trigger* levels. Activities are rescheduled to satisfy the trigger levels until a set of good schedules is obtained; then the best of them is chosen.

Resource allocation models have interested operations researchers for more than twenty-five years. Patterson [79] studied three different optimization models: (1) a bounded enumeration approach [21]; (2) an implicit enumeration method [97]; and (3) a branch-and-bound method [96]. His results, based on 110 projects, indicate that the branch-and-bound method of Stinson et al. provides the best results. Other optimization methods have similar approaches: the implicit enumeration algorithm of Patterson and Groth [78], the branch-and-bound method of Willis and Hastings [113] and the dynamic programming method of Petrovic [81].

The effectiveness of heuristic methods has been studied by Davis and Patterson, who examined eight different heuristic methods for 83 multi-resource constrained projects. They observed that [23, p. 952]:

None of the heuristic rules tested performed consistently best on all eighty-three problems. However, the MINSLK rule [Minimum Slack rule], which bases activity priority on activity slack, produced an optimal schedule span most often and exhibited the lowest average increase above optimum of the rules examined.

Kurtulus and Davis [60] studied the performance of heuristic resource-constrained methods for several projects. Their study indicates that to be effective, resource allocation algorithms should vary the heuristic rules they use depending on the status of the scheduling process:

... it is assumed that once a rule is selected it must be used throughout the whole project. This research breaks away from this tradition by providing a categorization process based on two powerful project summary measures. The first measure identifies the location of the peak of total resource requirements and the second measure identifies the rate of utilization of each resource type.

In light of this, knowledge-based procedures may be a useful avenue to explore for this problem.

Most resource allocation models ignore the relationship between resources and activity durations. They assume that durations are fixed, and that the goal is to appropriately allocate the available resources. However, some researchers

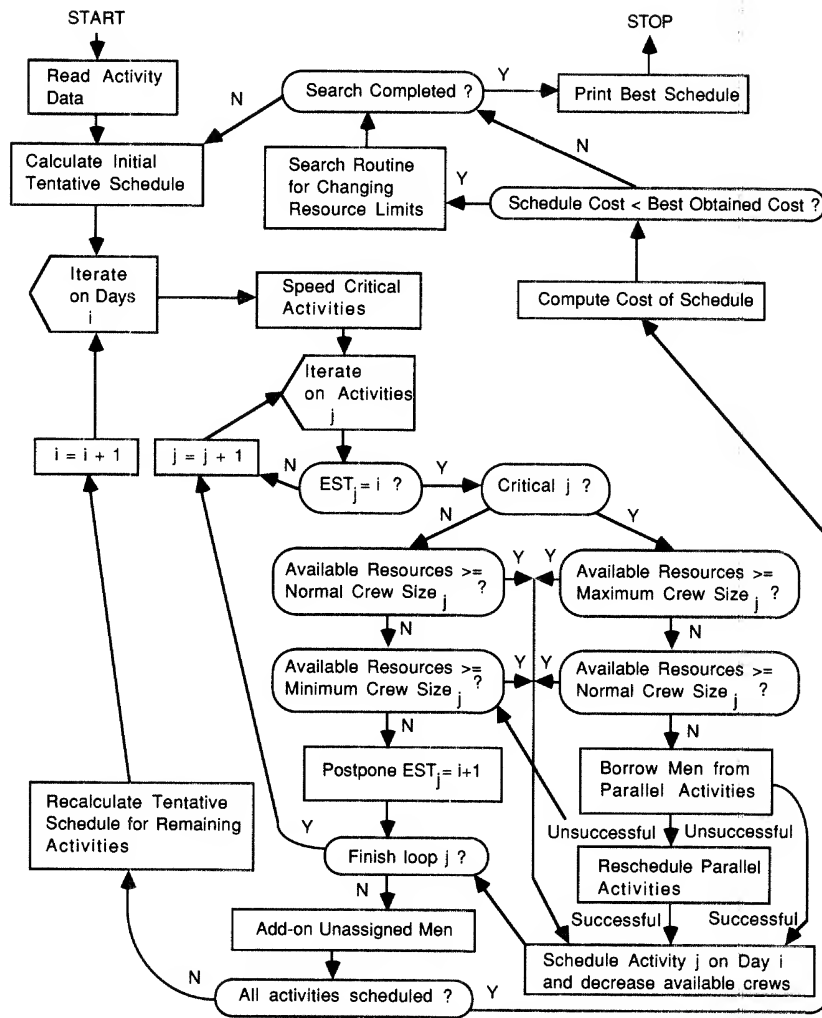


Figure 2-19. Simplified Flow-Chart of the SPAR-1 Heuristic Model

have explored the solution of resource allocation models in which activity durations depend on the type and number of resources allocated to the activities. Examples are the integer programming formulation of Elmaghraby [26, p. 173] and the model of Talbot [98]. An important heuristic model that addresses this problem is SPAR-1 (*Scheduling Program for Allocation of Resources*) [109]. The original version of SPAR-1 was capable of handling multiple alternatives for crew sizes and resource profile limits. Figure 2-19 (adapted from [109])

shows a simplified flow-chart of the SPAR-1 model. A brief description of the model follows:

- If an activity requires more than one resource, it is divided into separate activities that are constrained to start on the same day, one for each resource.
- Associated with each activity is data describing three crew sizes corresponding to the maximum, normal and minimum number of men in the crew for the activity. If required, the model may assign any crew size within the minimum and the maximum bounds.
- Activities to be scheduled are sorted in ascending order by their earliest-start-time and their total slack. Slacks and earliest-start-times are recomputed dynamically during the scheduling process.
- Activities are scheduled sequentially on a day-by-day basis, trying to shorten critical activities.
- If there are insufficient resources to schedule a critical activity, the system tries to borrow resources from non-critical competing activities already scheduled.
- If the borrowing process is unsuccessful, the system tries to reschedule activities without delaying the project. If this rescheduling process is unsuccessful, the system modifies the earliest-start-time of the activity being scheduled.
- When extra resources are available on a particular day, the system temporarily assigns additional resources to the activities scheduled on that day in ascending order of their total slack.
- Schedules are generated for different combinations of resource profile limits. An outer loop contains a search routine to adjust the total resource profile limits depending on the attributes of the schedules produced.
- Schedules are evaluated using a cost function that combines due-date penalties, resource costs, overhead costs and additional costs related to resource level changes.

2.2.5 Models for Combined Planning and Scheduling

An early attempt at introducing planning decisions within project scheduling methods was the Crowston and Thompson formulation of the *Decision CPM* (DCPM) [16, p. 407]:

DCPM is a method for formally considering the interaction between the scheduling and the planning phases of a project. Thus, if there are a number of competing methods of performing some of the jobs, each method having a different cost, a different time duration, and different technological dependencies, these possibilities are included in the project graph.

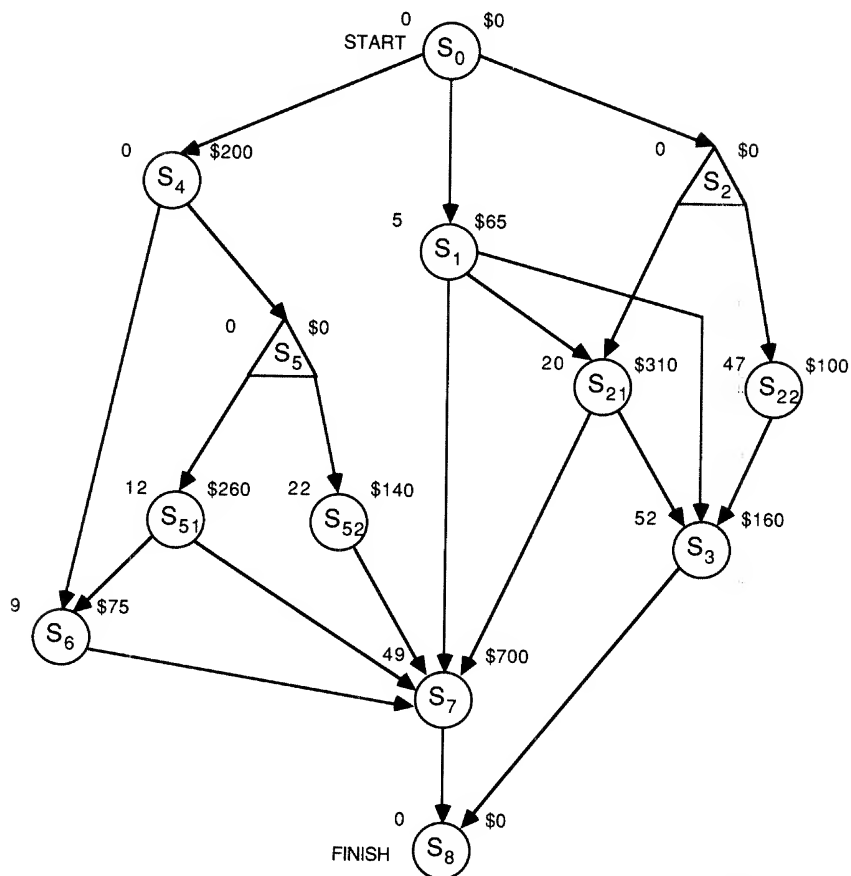


Figure 2-20. Example of a Decision Project Graph

In the DCPM model, planning decisions are represented by inserting decision nodes into an AON project network. The result is a *decision project graph* that includes activity nodes as well as decision nodes. Figure 2-20 (adapted from [16, p. 412]) shows an example of a decision project graph containing two decision nodes, S_7 and S_2 . These nodes are used to indicate different strategies for performing tasks. For example, decision node S_2 indicates that activity two can be performed in two different ways: one involving a cost of \$310 and a duration of 20 days, and one involving a cost of \$100 and a duration of 47 days. Furthermore, if the activity is performed using the \$310 alternative, it must precede the execution of activities S_7 and S_3 . However, if it is performed using the \$100 alternative, it need only precede activity S_3 .

The original DCPM method used a heuristic, iterative procedure to solve the problem. In this procedure, decision nodes on the critical path are revised until no further reduction in the overall cost of the project is obtained. The main steps of the DCPM method are:

- Step 1. Topologically order the activities.*
- Step 2. Identify and evaluate the set of choices.* The set of choices is formed by obtaining feasible combinations of individual node choices (i.e., which branch to follow). Each member of the set represents a particular combination of values for the decision nodes. This produces a set of networks, each with an associated overall cost.
- Step 3. Select the lowest cost combination in the set of choices.* After the lowest cost network has been selected from the set of choices, values (branch choice in the selected network) are assigned to the decision nodes.
- Step 4. Calculate the critical path and reorder the activities on the basis of earliest-start-time.*
- Step 5. Look for improvements.* The algorithm determines whether changes in the decision nodes on the critical path can reduce the overall cost of the project. This is done by analyzing the slack variables in all parallel chains going from the node under consideration to the terminal node. If no improvement can be made, the algorithm terminates.
- Step 6. Modify decision nodes.* The values of the decision nodes are changed to the alternative with maximum cost reduction. This results in a new network.
- Step 7. Recalculate the critical path.* The critical path in the new network is computed. The algorithm continues from Step 5.

The original DCPM method has been modified to improve its efficiency. Alternatives such as branch-and-bound methods and network reduction techniques were explored [17]. However, the potential of the DCPM method was extended with the development of an efficient dynamic programming algorithm by Hindelang and Muth [52]. In discussing the advantages of this algorithm, the authors state [52, p. 240]:

... the dynamic programming algorithm has proven quite powerful and, indeed, successful in overcoming the shortcomings of the previous approaches. Namely, computer time grows only about linearly with the number of arcs between nodes, and memory requirements are rather modest. Hence, large practical problems can now be accurately represented by the model and solved efficiently utilizing the new algorithm.

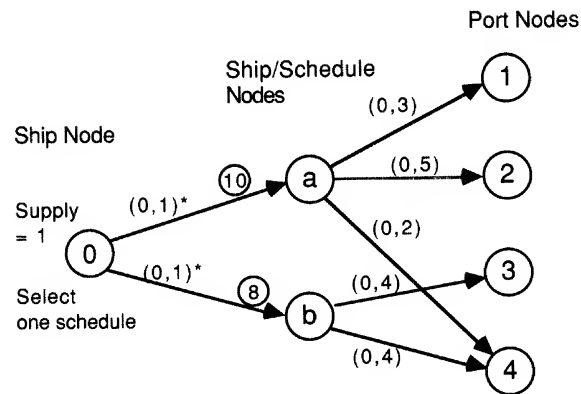


Figure 2-21. Example of a Generalized Network

Another model that may be used for combined planning and scheduling is a network formulation model, called NETFORM, proposed by Glover and Klingman [39, 40]. The authors illustrate how the NETFORM model can be used to solve *pure* and *non-pure* network problems using *Generalized Networks* (GN). Pure network problems include shortest path (CPM/PERT), assignment, transportation and transshipment problems. Non-pure problems include integer and mixed integer programming problems. The model uses efficient network algorithms to solve both types of problems.

An example of a GN is shown in Figure 2-21 (adapted from [40, p. 14]). Arcs of a GN contain information not found in pure network models [58]. Besides an associated cost and flow bound, each arc has a multiplier that indicates changes in flow magnitude along the arc and may have an asterisk specifying that the flow in the arc must be an integer. For example, in the GN of Figure 2-21, flow in arc $(0, a)$ is restricted to be either 0 or 1. This GN was used to model a problem of choosing delivery schedules for ships, as explained by Glover, Hultz and Klingman [40, p. 14]:

The setting for the example of [Figure 2-21] is a ship scheduling problem. In general, such a problem would involve many ships, a variety of schedules for each, and numerous ports (each represented in several different time periods). Here we show the part of the model that applies to a single ship with exactly two schedules, A and B. Schedule A requires the ship to carry 10 tons of ore, which is distributed among the ports by dropping 3 tons at Port 1, 5 tons at Port 2, and 2 tons at Port 4. Schedule B requires the ship to carry 8 tons of ore, dropping 4 tons each at Ports 3 and 4.

The potential advantages of applying this type of model to solve scheduling problems are discussed by the authors [40, p. 15]:

A recent application of the model, which schedules Air Force pilots to advanced flight training courses [...], illustrates the power of specialized methods for such classes of NETFORM's. The standard mathematical programming formulation of this problem is a 0-1 integer programming (IP) problem with 460 0-1 variables and 520 constraints. An attempt by the Air Force to solve this problem with an IP solution routine was abandoned due to the prohibitive amount of computer time consumed in the solution effort. By contrast, a branch-and-bound approach specialized for the NETFORM (solving GN subproblems) normally obtains and verifies optimal solutions within 30 seconds on a CDC 6600.

Thus, several methods exist for combining planning and scheduling in the framework of optimization. However, these methods assume that all activities and possible decisions have been previously identified and formulated in network models. Moreover, only a limited number of planning decisions can be included in the models due to computational complexity. As a result, optimization methods serve only a limited role in process planning.

2.3 Construction and Manufacturing Planning

This section reviews literature related to different elements of the construction and manufacturing planning process. Given the broad scope of the area, this review focuses on the material that affects the manner in which planning is currently performed. Previous work is discussed in relation to the elements of the construction or manufacturing planning process with which they are most closely associated. Various domain-specific expert systems are also reviewed.

Construction and manufacturing planning can proceed at many different levels of detail and abstraction. For manufacturing processes, typical levels of planning might be:

- *machining* in which the activities and resources required to produce a single part are planned;
- *assembly* in which the processes required to manipulate and join components are planned; and
- *production planning* in which resource assignments and schedules are developed for an entire manufacturing facility.

Each of these levels includes fundamental problems of plan generation and evaluation. Descriptions of systems for process planning for particular types of tasks appear in Chang and Wysk [11].

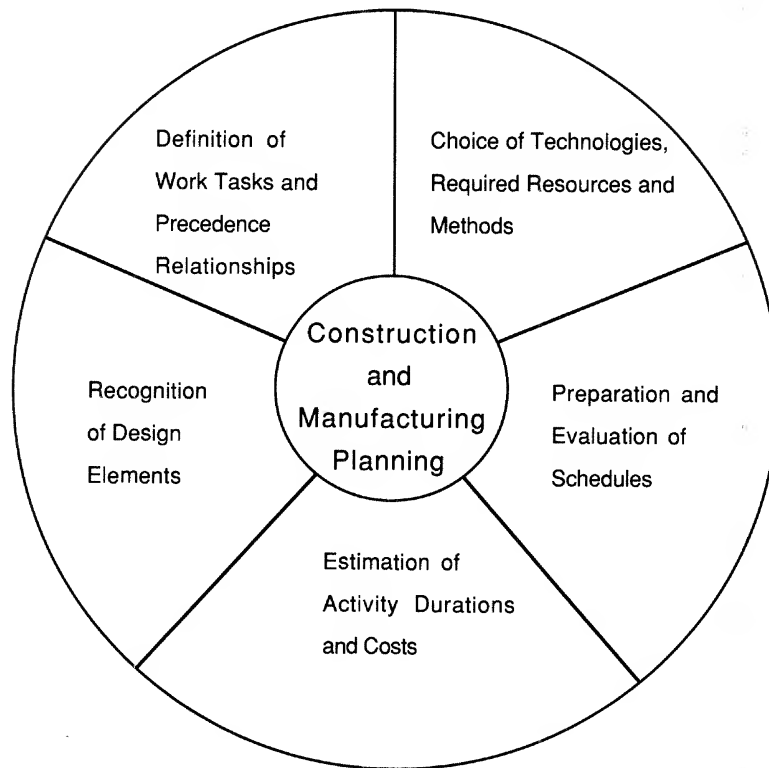


Figure 2-22. Elements of the Construction and Manufacturing Planning Process

Figure 2-22 shows the general elements or steps of the construction and manufacturing planning and scheduling process:

- *Recognition of Design Features and Elements* involves the extraction and classification of important elements or features in the final design;
- *Definition of Work Tasks and Precedence Relationships* involves defining the activities to be performed and identifying any technological or resource management precedences among these activities;
- *Choice of Technologies, Processes and Resources* involves decisions about appropriate technology and methods for performing tasks;
- *Estimation of Activity Durations and Costs* is the determination of the expected durations and costs of the activities on the basis of the technology and method selected for performing each activity; and
- *Preparation, Evaluation and Maintenance of Project Schedules* involves generating project schedules that satisfy the resource, time and cost constraints imposed on the activities, and then analyzing these schedules with respect to managerial goals.

These elements of the planning process are interdependent because executing any one of the steps eventually affects the execution of the others. Therefore, some of these steps should be performed in parallel or in loops with back-tracking.

2.3.1 *Recognition of Design Features and Elements*

Before beginning process planning per se, a representation or model of the design problem must be created. This model must contain a taxonomy or nomenclature to permit subsequent analysis.

In manufacturing, considerable work has been done to devise general coding systems to describe parts and features under the general subject of *group technology*. Typically, coding systems for group technology are hierarchical, matrix or a combination of these methods (Figure 2-23). In hierarchical structures, digits represent the appropriate descriptor at each level of the hierarchy. Matrix or chain codes represent table look-up instructions. Hybrid systems include both of these representations. For example, a nine digit code for a physical component using the Opitz system [77] might use the first digit to denote part class, a second digit for main shape, a third for rotational machining type, a fourth for planar surface machining type and a fifth digit for additional holes, teeth and forming indicators. Additional digits would denote dimensions, material, original shape and required accuracy.

Recognition of design features for the purpose of manufacturing may require considerable domain-specific knowledge [46]. Numerous design features with seemingly different attributes can actually be produced by the same manufacturing processes. Moreover, aggregation of design components into sub-assemblies may be done to improve manufacturing efficiency without specific guidance from the design specification. Similar problems arise in construction in aggregating design elements into work sectors.

2.3.2 *Definition of Work Tasks and Precedence Relationships*

Most literature related to the definition of work tasks in construction and manufacturing emphasizes the importance of defining work tasks and precedence relationships in the planning process and discusses the characteristics of appropriate *work breakdown* (WB) structures. The process of obtaining adequate WB structures for construction and manufacturing has been studied only recently. This section presents two models that can be used to identify project activities. Programs that generate activity networks for projects are also described.

Halpin et al. [44] proposed a WB model intended to be consistent in the design, procurement and construction phases of a project. Their model uses a hierarchical decomposition of project tasks along three different perspectives:

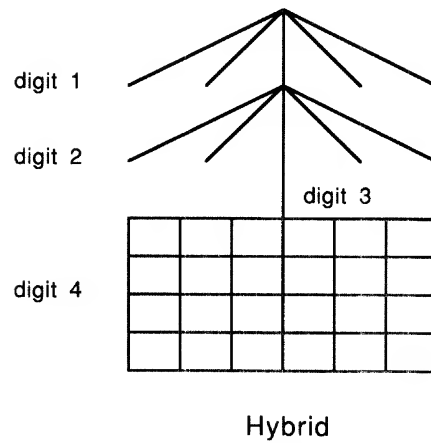
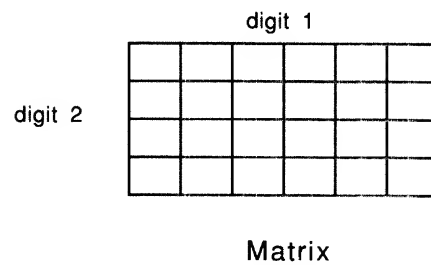
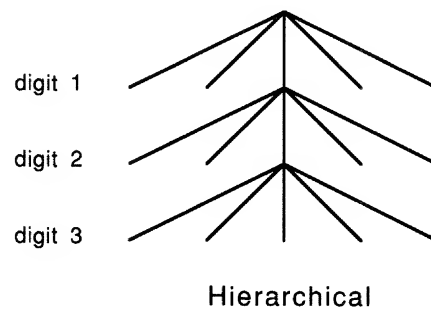


Figure 2-23. Illustration of Hierarchical, Matrix and Hybrid Codes

- the *physical* perspective describes the project in terms of major end items, systems and components;
- the *organizational* perspective relates work tasks to the part of the organization responsible for the task; and
- the *resource* perspective groups work tasks with respect to the type of resources they use.

The physical perspective decomposes the project into very simple operations for which time and cost estimates can be made accurately. The other two perspectives map these detailed operations into other dimensions for managerial and reporting purposes. Having decomposed the project using these perspectives, *work packages* are defined by grouping the simple operations. The purpose of this aggregation is to produce elements of the project that are meaningful for scheduling and monitoring purposes. In the model, each work package is described by a unique code that specifies its composition with respect to the three perspectives. Work package descriptions are stored in a dictionary that is used for cost and control purposes.

A second approach is offered by Baracco's model for integrating duration and cost estimating [4]. The model addresses deficiencies in the current methods used by construction planners in preparing budgets and schedules for building facilities. He identified some deficiencies in current practice and developed a model for integrating duration and cost estimating. An overview of the model is shown in Figure 2-24. The estimating process starts with information about the *design* of a facility and the *site* where the facility is to be constructed. Design information includes *plans* and *specifications*. Plans consist of drawings of different systems or areas of the building, such as electrical systems, mechanical layouts, structural components or functional space areas. Specifications provide information about materials, finishes and other design data not included in the plans. Site information contains data on the construction site that affect the construction process, such as soil composition and climatic conditions.

Using the information in plans and specifications, the planner decomposes the building into *design elements*. A design element is a unitary component of a facility, such as a column, a beam, a column footing or a slab. Then the planner identifies *elements-of-work* for each design element. An element-of-work is an activity that is performed to construct a design element. Examples of elements-of-work are the formwork and concrete pouring activities required to construct a concrete column. In the model, elements-of-work are identified by a unique code that extends the standard hierarchical MASTERFORMAT code [18] by adding information about the type and location of the design element associated with the element-of-work. The purpose of this code is similar to that of Halpin's model: to provide a unique identifier for aggregating information along various dimensions.

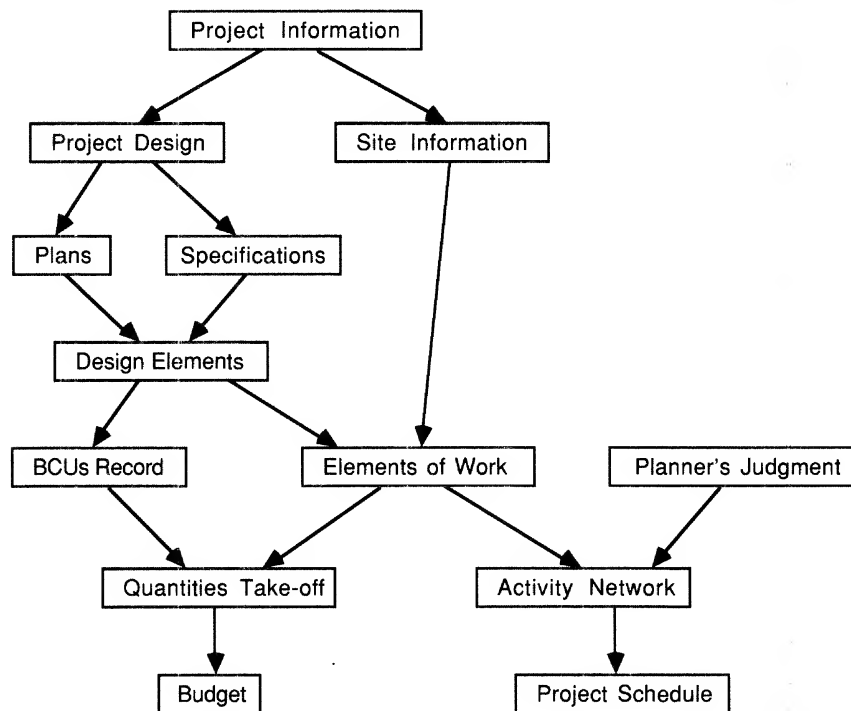


Figure 2-24. An Integrated Construction Planning Model

The importance of elements-of-work in Baracco's model is that they provide the basis for creating a project activity network and a cost estimate based on the amount of work or quantity take-offs. Cost estimates are computed when the planner selects an appropriate *Basic Cost Unit* (BCU) for each element-of-work. Each BCU is a package of labor, material and equipment that may be used to perform the element-of-work under certain conditions. The direct cost of an element-of-work is obtained by dividing the amount of work by the standard productivity of its associated BCU and multiplying by the hourly cost of the BCU. Project activities are created by aggregating elements-of-work. Once project activities are created, the planner determines precedences among them, thus producing an activity network that is used for scheduling and monitoring purposes.

Recently several researchers have studied the problem of automatically creating the project activity networks. One system, called GHOST (*Generator of Hierarchical schedules for cOnStruction*) [75], receives as input a set of activities and finds precedences among these activities by using a set of *critics*. GHOST starts with a list of unstructured activities such as *build lintel* or *build*

foundation and creates a network in which all activities are performed in parallel. Then it applies critics to find physical precedences among the activities by using knowledge about construction and physical relationships. Once a network has been created at a particular level of detail, the system expands each activity into a set of subactivities. Critics are applied to the newly created network to find precedences at this lower level of detail. The system proceeds by successively expanding networks until an appropriate level of detail has been obtained. The limitations of the system are:

1. GHOST generates too much redundancy during the hierarchical inheritance;
2. it does not check for circularity;
3. it does not interpret actual construction drawings;
4. GHOST cannot estimate activity durations;
5. it needs to be able to handle global pre-activities;
6. it cannot schedule the trades that will be involved in the project; and
7. it cannot redefine the dependencies among subactivities.

Despite these limitations, GHOST is an interesting application of a planning model which functions similarly to NOAH's process of successively expanding nonlinear plans (see p. 23). In addition, the list of initial activities input to GHOST is similar to the list of design elements of Baracco's model for work decomposition. Each design element (e.g., *pillar A*) is associated with an aggregate activity representing the process of building this design element (e.g., *build pillar A*). Then these activities are expanded into subactivities that resemble Baracco's elements-of-work (e.g., *build formwork* or *place reinforcement*). More powerful systems of this type are appearing, such as PIPPA [67] and OARPLAN [20].

Another system, called LIFT-2 [7], is a rule-based expert system that supports the modeling of heavy lifting operations. LIFT-2 takes the role of an engineer in a contracting firm who is preparing a bid for offshore heavy lifting jobs. For this task, the planner has to develop a plan for the lifting job by analyzing the geographic characteristics of the site, the location of available cranes and the location of platform modules. LIFT-2 produces a graphical outline of the plan as an AON representation and provides explanations of the reasoning process. In solving the problem, LIFT-2 combines different problem-solving strategies. The overall reasoning strategy sequentially satisfies goals in an attempt to effectively reduce the total search space. For example, the goal *vessel selection* is satisfied before the goal *site investigation*. Goals are satisfied using a *generate-and-test* paradigm: when trying to satisfy a goal, all possible alternatives are generated and some of them are eliminated by using previously formulated constraints. The justifications and assumptions for using this paradigm in the higher levels of the problem-solving process are described as [7, p. 32]:

Constraint satisfaction using the generate-and-test-paradigm is successful at the highest levels in this hierarchy because sub-goals are assumed independent and because constraints and generators (i.e., user-prompted input) are available. In some cases, however, where sub-goals at the highest levels are not completely isolated this paradigm is weakened.

Other problem-solving techniques used in LIFT-2 are *least-commitment* and *meta-planning*. Least-commitment is used in the lowest levels of the problem-solving process. At these levels, the interactions among subgoals are important and the system deals with them by using rules that combine possible solutions to several goals. This strategy delays decision-making until enough information is available to solve several subgoals simultaneously. Although the system does not have a well-defined layered structure as in MOLGEN (see p. 30), the rules in LIFT-2 contain both strategic and detailed knowledge.

In LIFT-3 [8], a descendent of the LIFT-2 system, the planner is not part of the lifting firm, but works for the oil company responsible for the complete project. The architecture of LIFT-3 uses the blackboard problem-solving paradigm of OPM (see p. 32) and replaces the sequential reasoning process of LIFT-2 with an opportunistic process in which independent knowledge sources are executed. In addition, LIFT-3 reduces the search space by identifying ranges for the values of the design variables rather than committing itself to assigning values to these variables.

A final example of a program that generates activity networks in TIPPS (*Totally Integrated Process Planning System*). TIPPS evolved from the earlier APPAS and CADCAM programs [11] and contains a process selection module that defines machining activities and selects manufacturing processes simultaneously. Appropriate processes to achieve desired design features are selected by evaluating decision tables. A typical process knowledge data element is:

```
(IF (SHAPE ! xxx =)) (THEN (8 PROCESS 8 ))
```

which is interpreted as: feature xxx should use process 8.

2.3.3 Choice of Technology and Method

Selecting the technology to perform tasks involves two types of decisions. First, the planner has to identify and choose among possible packages of labor and equipment available to perform the task. Alternate methods affect the resource requirements. For example, Walker's estimating book [105, p. 8.117] indicates various personnel requirements for mixing and placing concrete in foundations that are based on the chosen construction method (see Figure 2-25).

In selecting the type of technology package or crew, the planner uses knowledge about crews or equipment usually chosen to perform an activity. For construction activities, this information is available in construction estimating books [105], in books about construction methods [76] or, more generally, in

Construction Method	Hours per Cubic Yard	
	Mixer Engineer	Foreman
Hand Mixing (one board)	None	0.33
Hand Mixing (two boards)	None	0.25
One Sack Mixer	None	0.17
Two Sack Mixer	0.125	0.125
1 Cu. Yd. Mixer	0.083	0.08

Figure 2-25. Example Unit Requirements for Mixing and Placing Concrete in Foundations

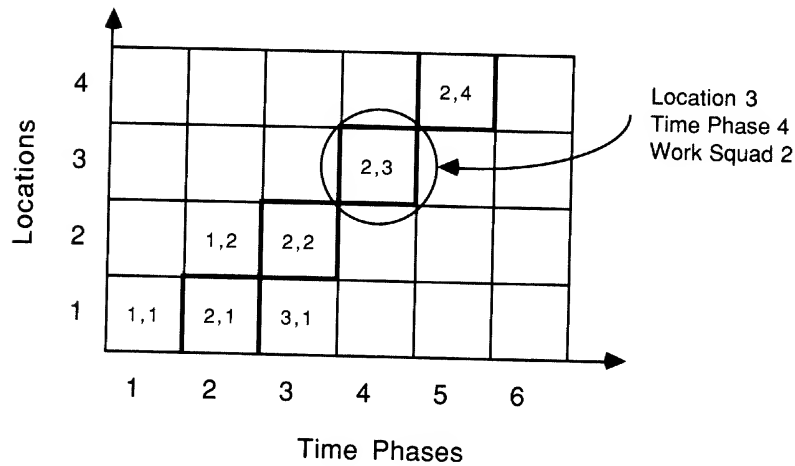


Figure 2-26. Illustration of Line Balancing

manuals that list appropriate crews for construction activities, typically utilizing the MASTERFORMAT coding system [71].

After the type of crew or technology package for an activity has been chosen, the second type of decision made by the planner is to select the number of machines or crews assigned to the activities. There are two alternative methodologies that may be used in this decision process. In the first alternative, the planner knows an approximate value for the duration of the activity and adjusts the number of crews allocated to the activity in order to achieve this duration. This approach is based on experience from previous projects and does not consider the interactions between the activities.

In the second alternative, the planner simulates the manner in which the different crews and machine types will be used to perform the activities and tries to achieve a continuous use of the crews. In this approach, usually called *assembly line balancing*, the objective is to insure that each activity proceeds at

the same rate of speed. For example, Figure 2-26 illustrates the progression of a component or a work task through different time phases and through different equipment locations. In a balanced assembly line, subsequent components would not be delayed at any stage of the manufacturing process.

Another technique that is used to aid technology selection is process simulation. An example is CYCLONE [42] (*CYCLic Operations Network*), which has been successfully used to model the construction of several buildings, including the Peachtree Center Plaza Hotel in Atlanta [43]. Figure 2-27 (adapted from [42, p. 69]) shows an operations network for modeling earthmoving operations. The network contains different types of elements:

- *Combination activity nodes* represent activities that cannot be performed until a combination of required resources is available. If only some of the resources are present, the activity waits for the arrival of the remaining resources before it commences. An example is the *load truck* activity that requires an idle *front-end loader*, a non-empty *truck queue* and a *soil stock pile* with enough soil before the activity can start.
- *Normal activity nodes* represent activities that may be executed whenever any of the required resources are available. Resources used by the activities do not have to wait before they are used. An example is the *haul to dumping area* activity that may begin as soon as the *load truck* activity finishes.
- *Queue nodes* represent places where labor, equipment or materials wait before being used by an activity. Examples are the *truck queue* and the *front-end loader idle* nodes of the network.
- *Arrows* represent flows of resources (e.g., soil, trucks or loaders).

In addition to these elements, CYCLONE networks may include *function nodes* and *accumulator nodes*. Accumulator nodes are used to control the simulation process and to obtain statistics on the number of resource units flowing through an arrow element. Function nodes are used to introduce user-defined functions such as those that define resource availability.

The CYCLONE model and other process simulation tools are powerful aids for evaluating alternative technologies or methods. However, these models require the user to manually define the operations network to be analyzed. CYCLONE has no provisions for modifying the topology of the operations network during the simulation or after the simulation has been completed. Therefore, CYCLONE is considered more of an *analysis* aid rather than a *synthesis* aid for the planner.

Another model that could be used to select among alternative resource choices is the Decision CPM model (see p. 44). In this model, decision nodes are used to represent combinations of crew types and number of crews. For example, an activity with two possible types of crews and three alternative crew sizes (e.g., maximum, desirable and minimum) is represented using a decision node with six possible outcomes.

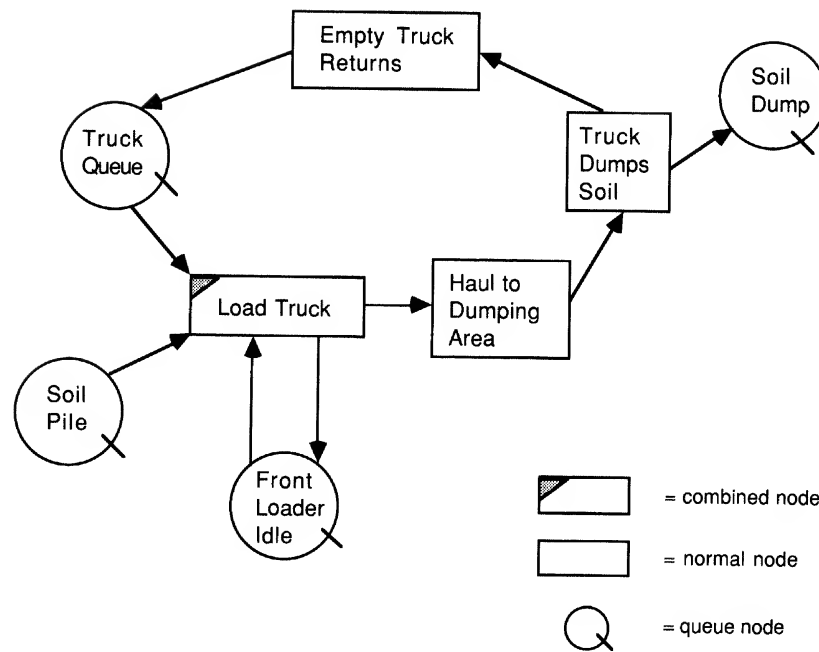


Figure 2-27. CYCLONE Network for Modeling Earthmoving Operations

A final example of a technology selection aid is graphical process simulators in which planners can view an animation of process activities and equipment on the plant floor. Again, these models are more useful for analysis and visualization than for process synthesis.

2.3.4 Estimation of Activity Durations and Costs

Estimation of activity durations is a fundamental task in the planning process. However, it has received little systematic attention [90]. Project planners usually base their estimates on average productivities found in estimating handbooks such as Dagostino's [19] or in company records. Adjusting these productivities to reflect local conditions requires considerable expertise and thus cannot be done successfully by novice planners.

Researchers have attempted to capture the expertise of planners by using expert systems technology. One of these systems, MASON [48], is used to estimate the duration of masonry construction activities. The system has explanation capabilities and provides the user with recommendations for improving crew productivities. In solving the estimating problem, MASON uses a hierarchical, rule-based estimation process in which higher levels represent at-

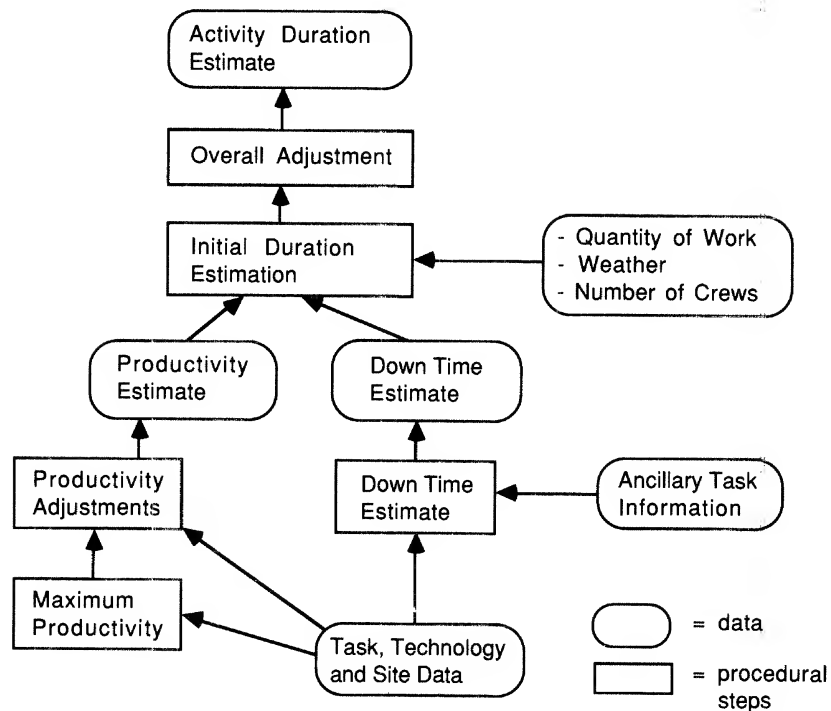


Figure 2-28. MASON's Estimation Hierarchy

tributes which depend upon the details of lower-level inferences and calculations. The estimation hierarchy is represented in Figure 2-28 (adapted from [48, p. 291]). At the lowest level of the hierarchy, MASON estimates the maximum crew productivity and identifies various modifications to account for downtime (i.e., idle, waiting). At the next highest level, adjustments to productivities are computed by considering special characteristics of the site or of the job. At the top level, quantity of work, productivity, crew resources and downtime are combined to estimate the duration of the activity.

Similarly, planning manufacturing processes also involves estimating to select equipment parameters. For example, the following rule is used to identify the number of passes and material speed for a machining process in the PROPLAN planning system [74]:

If <Operation is Turning>
<Height of REC+ is between 1.5 and 2 inches>
<Partmaterial is Cast Iron>
<Toolmaterial is Carbide Tip>
<Surface Finish Required is Regular>

Then <Speed = 250 RPM>
<Rough Passes = 5>
<Finish Passes = 2>

2.3.5 *Preparation, Evaluation and Maintenance of Project Schedules*

Process planning may be divided into two major phases: (1) a preconstruction or premanufacturing phase; and (2) a process phase. Some of the tasks performed by the planner in the first phase, such as activity duration estimation or technology choice, have already been described. After performing these tasks, the final step in the preconstruction and premanufacturing phase involves the preparation of a work schedule. In obtaining this schedule, planners usually employ one or several of the basic scheduling techniques described in Section 2.2. Common techniques are the Basic CPM method, the Precedence Diagramming Method, or some resource allocation model with fixed heuristics for *job shop scheduling* [3]. Planners use these techniques to identify the critical activities, compute the earliest-start-time of the activities and their corresponding slacks, estimate the overall duration of the project, and perform litigation during the bidding phase. Some contractors or manufacturers use scheduling techniques during the construction or manufacturing phase of a project [1]. However, existing computer tools do not incorporate the knowledge required to solve the problems encountered in the project monitoring process. Potential applications of knowledge-based expert systems in this area are discussed by McGartland and Hendrickson [69]. In this section, some computer systems that may be used to assist the planner in maintaining and evaluating project schedules are described.

An interesting approach for incorporating construction knowledge into project scheduling systems is illustrated by Levitt's PLATFORM [62], a prototypical expert system for automated updating of activity schedules for design and construction. The domain is concrete, offshore, oil drilling platforms. PLATFORM represents activities by using frames with slots that contain static information (e.g., optimistic and pessimistic durations) and procedures for computing other activity attributes (e.g., expected duration and variance). This representation allows detailed activities to inherit properties of parent frames corresponding to more general activities. In addition, activity frames contain

information about causes or impacts that may affect the durations of the activity, such as the designer's competence or the owner's requirements.

PLATFORM updates the initial project schedule by recording the actual duration of completed activities and by identifying *knights* and *villains*. Knights represent characteristics common to two or more activities whose actual durations were shorter than expected. Villains represent characteristics common to two or more completed activities whose durations were longer than expected. Knights and villains are used to update the estimated duration of the activities not yet completed. Knights change the expected duration of an activity to its optimistic duration and villains set it equal to its pessimistic duration. In the updating process, the user may override the system's recommendations.

Although the main purpose of PLATFORM was to illustrate the use of AI techniques for schedule maintenance, the system also permits limited changes to network topology by explicitly storing alternative subnetworks for each major activity. For example, depending on the soil condition (e.g., sand versus clay), the system selects one of these subnetworks when creating the final project network.

A major development in the creation of intelligent project management systems was CALLISTO [88]. CALLISTO was initiated as an effort to produce an intelligent assistant for use during the design and prototype development stages of large computer systems. The CALLISTO project has produced several prototypical systems that illustrate the potential of integrated project management systems. The first prototype was rule-based and focused on a single project plan that was constantly monitored and updated. This prototype was used to explore the types of expertise used in the scheduling process, but it assumed that the different parts of the organization shared common goals. Later this concept evolved into a new architecture that emphasized the need for negotiation among different organizational units. In the new architecture, each organizational unit is represented by a *Mini-Callisto* system that incorporates its own expertise and goals. These Mini-Callistos interact and negotiate a solution that satisfies their individual goals as well as the goals of the project.

In CALLISTO, the project management problem is decomposed into three major activities:

- *Activity Management* involves the planning and scheduling of activity networks and the evaluation and monitoring of activity schedules;
- *Resource Management* involves the projection, acquisition, assignment and maintenance of resources; and
- *Configuration Management* involves the management of product specifications and changes.

These three activities are very similar to those performed by contractors. Contractors not only deal with activity management issues, but also perform many

other tasks related to the management of resources and specifications. Contractors have to purchase materials and lease or buy needed equipment. Also, they must negotiate with the owner over work change orders.

Usually the many tasks involved in project management are performed by different parts of an organization. Therefore, the Mini-Callisto model of negotiations between organizational units seems to be a promising approach in developing a complete project management system. PLANEX, however, focuses on the planning and scheduling problems of the activity management area. The methods employed by CALLISTO for representing activity knowledge [87] provided insights that were used in designing the knowledge-based system architecture of PLANEX.

2.4 Conclusions

This chapter has reviewed previous work in three main areas: *plan formulation*; *project scheduling*; and *process planning*. Plan formulation systems focus on identifying sequences or networks of activities used to meet certain goals. Most of these systems do not involve complicated considerations of time, budget or resource constraints. In addition, their application has dealt with toy domains (e.g., those related to stacking blocks with a robotic hand) rather than with more realistic problems. Conversely, project scheduling research has concentrated on developing efficient algorithms for incorporating resource, time and budget constraints into project management models. However, most of the scheduling systems do not consider changes in the topology of the activity network or in the type of resources assigned to each activity. Therefore, they may be used only *after* the planner has made many of the decisions in the planning process. Finally, reviewing the tools specifically developed for managing projects reveals that planners receive little help from existing models. Most of the planning process is done by hand and important tasks such as schedule updating are not performed regularly [62, p. 57]:

Project managers and senior estimators are typically unwilling or unable to devote large blocks of time to maintaining schedules for real time planning and control purposes during a project. Thus, schedule updating becomes primarily an archival record-keeping process, rather than a replanning process, and is carried out by lower-level scheduling engineers.

Some probable reasons for the minimal impact of existing models in the planning process are:

- *Systems focus on analysis rather than on synthesis.* Existing resource selection packages still require the planner to identify which resources may be used to perform the activities. Research has focused on simulating the selection of technology choices previously done manually. There is a need for tools which can assist during the decision-making process.

- *Systems do not incorporate domain-specific knowledge.* Effective aids for the planning process must incorporate and use task-specific knowledge. Commercial scheduling packages require the planner provide *data* about the activities (e.g., durations, costs, precedences, resources) but they do not incorporate *knowledge* about how the data was obtained. Only a few systems (e.g., PLATFORM) incorporate knowledge to assist during the revision of project schedules once the project has started.
- *Systems are not transparent.* Most systems are *black-boxes* because their internal heuristics are not known or cannot be changed by the user. Practitioners are not willing to use black-boxes because they do not understand them.
- *Systems are not flexible.* Planners require systems that are flexible and can be easily adapted to fit different planning scenarios and adjusted to meet different managerial goals.

Several surveys [22] and comparison studies [1] seem to indicate that the reasons listed above reflect the beliefs of practitioners. A knowledge-based planning architecture that is designed to overcome some limitations of existing systems is described in Chapter 4. This architecture builds upon the foundation of AI planning systems, optimization methods and knowledge-based planners reviewed in this chapter. Taking advantage of each of these approaches in an appropriate fashion was a major objective in developing the PLANEX system. A conceptual model suitable for this integration is described in the next chapter.

3 Modeling Process Planning Problems

The preceding chapter presented a review of previous work on planning systems. This review showed that automated planners differ considerably with respect to both their structural and behavioral characteristics. Recent planners have layered or blackboard architectures which are structurally more complex than the architectures of early planning systems. In terms of their behavior, some systems solve planning problems using algorithmic search procedures while others obtain plans using opportunistic or constraint propagation mechanisms. However, the major differences among automated planners are those related to the capabilities they provide for modeling planning problems. For example, means-end planners such as NOAH and NONLIN do not provide a mechanism for representing the combined effects of actions (e.g., the total usage of a resource). Therefore, these planning systems are not appropriate for modeling planning problems in which resource considerations are important (e.g., resource allocation problems). Thus, to develop a process planning system which incorporates all of the elements of construction and manufacturing planning, a conceptual model of planning process itself must be developed.

This chapter presents a conceptual model for process planning which may be used as the basis for designing automated process planners. The chapter begins by analyzing the type of information contained in process plans and the different entities involved in process planning problems. Following this analysis, the applicability of two AI planning models to the process planning problem is discussed and a hybrid model for process planning is described. Finally, the last section presents a list of architectural requirements for an automated system that would implement the conceptual process planning model. These requirements constitute the basis for the design of the knowledge-based process planning architecture presented in the next chapter.

3.1 A Conceptual Model for Process Planning

In this section a conceptual process planning model is described. This model combines elements of two previous planning models:

- the *means-end* planning model used by linear and nonlinear planners; and
- the *blackboard* planning model used by blackboard planners.

The rationale for developing such a model is explained by analyzing the difficulties that previous planning systems have in modeling process planning problems.

3.1.1 Characteristics of a Process Plan

Figure 3-1 shows a *black-box* representation of process planning. The planner analyzes the information describing the product to be built or manufactured and, based on previous experience, it generates a complete plan for creating the desired product using the set of available resources.

The information in a construction or manufacturing process plan is broader in scope than that produced by most AI planners because a process plan not only identifies which activities are required to build or manufacture a particular product, but also specifies other activity attributes such as durations, costs and resource utilization. In some domains, activities are linked with each other into a network that specifies the order in which activities must be performed (e.g., placing the forms for cast-in-place concrete must precede pouring the concrete). This network may be used in computing scheduling data for the activities such as their earliest or latest completion times.

Although the information contained in a particular process plan depends upon the characteristics of the corresponding construction or manufacturing process, a process plan always provides information about the relationships between three basic types of entities:

- *Product Components* which are portions of the final product to be constructed or manufactured;
- *Activities* which represent operations to be performed to produce the desired product; and
- *Resources* which are used in performing the activities, such as machines, labor or materials.

Typically, there are many combinations of activities and resources which may be used to construct or manufacture the final product. Thus, the planner faces a challenging situation because activities and resources are not independent. For example, in an excavation project, the set of available excavators affects the number and type of the excavation activities included in the process plan.

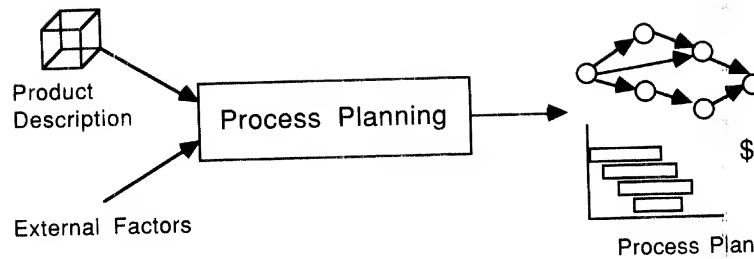


Figure 3-1. *Black-Box Representation of Process Planning*

3.1.2 Means-End Model

A first attempt at solving process planning problems would be to model them by employing the representation used by strategic means-end planners. Using this representation (see p. 16), the process planning problem would be formulated as:

- | | |
|--------|--|
| Given: | A description of a desired product
A set of available resources
A set of possible operations |
| Find: | A description of the manner in which processes and operations may be combined to obtain the desired product. |

However, this model of process planning presents two major problems:

- First, it is difficult to define the desired and initial conditions using a homogeneous terminology. Figure 3-2 illustrates this problem. Process planners have a clear idea of the product to be built or manufactured (e.g., a part or a building) because this information comes from the drawings and specifications of the product. However, it is difficult to describe the initial state (e.g., the beginning of the project) in the same terms as those used to describe the final product. The initial state is not a mere decomposition of the product into its primitive components (e.g., columns, beams), but more similar to a pool of materials, labor and equipment which are combined to obtain the final product.
- Second, it is difficult to specify the preconditions and effects of the means-end operators in terms related to the characteristics of the final product and available resources. For example, when NONLIN was used to plan construction projects, operators represented construction activities whose preconditions and effects were related to other activities and not to building components. An operator such as *pour-concrete-columns* would have a precondition of *finished-placing-forms-columns* and an activity-related effect of *finished-pour-concrete-columns*. This description does not provide meaningful information for process planning because there are no explicit relation-

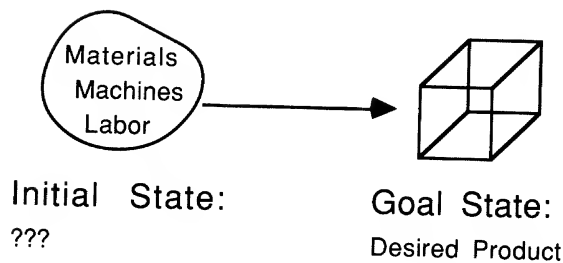


Figure 3-2. Process Planning as a Means-End Problem

ships with the characteristics of a column (e.g., cast-in-place concrete requires formwork). Furthermore, such a description of the operators requires knowledge of the activity network in advance.

Despite these problems, the concepts provided by a means-end model are useful for controlling the execution of process planning operators in the knowledge-based architecture described in Chapter 4.

3.1.3 Blackboard Model

Modeling process planning problems could be based on the blackboard model described in the previous chapter (see p. 32). Blackboard planners generate plans incrementally following a procedure in which multiple *agents* (e.g., the *knowledge sources* of OPM) contribute to developing portions of the final solution. Modeling process planning problems using the blackboard model requires:

- the identification of appropriate individual agents, which implies that the *black-box* representation of Figure 3-1 is decomposed into a set of simpler planning operations related to the different aspects of the process plan; and
- the specification of how these agents are controlled during the creation of a process plan, which implies that criteria for scheduling the planning operations are defined.

Figure 3-3 shows the first two levels of an illustrative hierarchical decomposition of process planning into simpler operations. In the first level, process planning is decomposed into four macro operations related to the elements of process planning discussed in Section 2.3. Each operation is successively decomposed into simpler operations until the desired level of detail is reached. The resulting set of low-level operations constitute the agents that are scheduled during the execution of the planning system.

There may be other mechanisms to obtain the set of planning agents in addition to the hierarchical decomposition procedure described above. In any case, the manner in which planning agents are obtained is not particularly

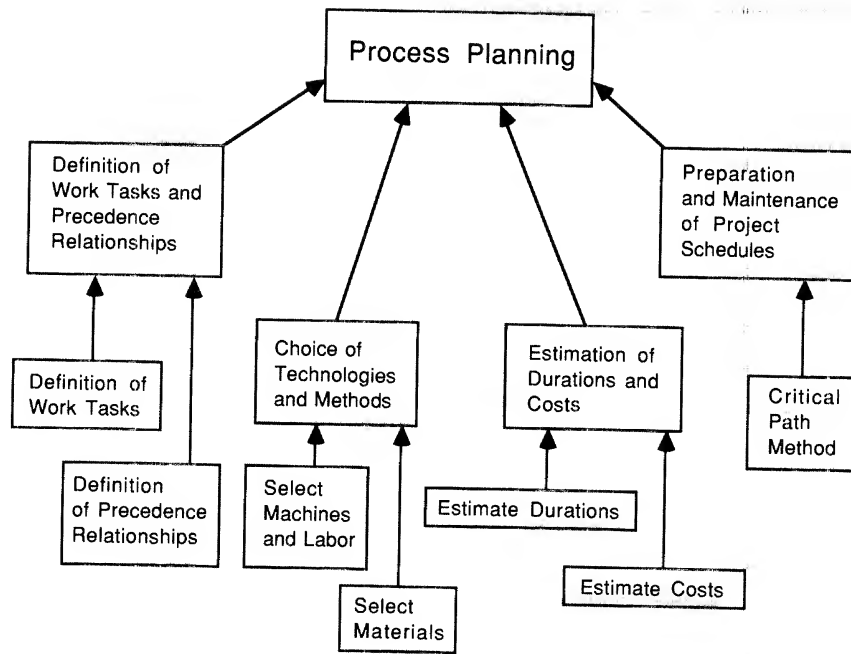


Figure 3-3. Example of a Hierarchical Decomposition of Process Planning

relevant to the execution of a blackboard planner. What is important, though, is to establish the manner in which these planning operators are scheduled during the problem-solving process. In order to use the blackboard model in a process planning system, it is not enough to specify that only one of the problem-solving agents is executed in each problem-solving cycle. Scheduling or conflict resolution criteria are needed to evaluate both the feasibility and desirability of executing the operators at any step in the planning process. Some blackboard planners such as OPM (see p. 32) have control agents which provide powerful control schemes. However, defining the scheduling criteria in specific domains is not easy because control knowledge depends on the nature of the planning operators themselves.

3.1.4 Hybrid Model

In order to overcome the difficulties that may arise when using either a means-end or blackboard model for process planning applications, a new planning model was developed. This model, called the *hybrid* model, combines modeling concepts from other planning models and was implemented in PLANEX.

Similar to the blackboard model, the hybrid model assumes that process planning can be decomposed into a set of simple planning operations, each related to a particular element of a process plan. However, the control structure of the hybrid model uses a declarative representation of operations which resembles the descriptions of the operators of a means-end planner. These descriptions suffice for controlling the execution of the planning operators, and no additional scheduling knowledge needs to be represented.

The behavior of a process planner using the hybrid model alternates between two levels of execution:

- a *strategic* level in which the planner behaves as a means-end planner and creates networks of planning operators that achieve goals or propagate changes using independent knowledge which describes each operator; and
- an *operative* level in which the planner executes operators one at a time, resembling the problem-solving cycle of a blackboard planner in that knowledge and procedures are decomposed into simple operators.

In addition, implementations of the model require an *interface* level to display results of the planning process and to allow the user to modify planning decisions.

Figure 3-4 shows the representation of a planning operator in the hybrid model. The operator has three lists of associated *conditions*:

- *preconditions* that must be true before the operator is executed;
- *positive effects* which are conditions asserted by the operator; and
- *negative effects* which are conditions negated by the operator.

These lists resemble the *preconditions*, *add* and *delete* lists of some means-end planners such as INTERPLAN and NONLIN (see Section 2.1). The difference, however, is that conditions are expressed in terms of the status of the *places* where information is stored or retrieved (e.g., whether or not the attribute is defined), rather than in terms of product or process plan characteristics (e.g., the geometric description of the desired product). For example, an operator that determines the duration of an activity using data about its *amount of work* would have a precondition indicating that the value of the *amount of work* attribute must be defined before executing the operator and an effect indicating that the duration of the activity will be defined after the operator is executed.

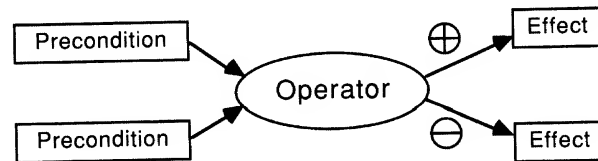


Figure 3-4. Representation of a Planning Operator in the Hybrid Model

Using the hybrid model, the feasibility of executing an operator is determined by looking at whether pieces of information (e.g., the attribute *X* of object *Y*) are available. However, the desirability of executing an operator can only be determined if goals are also expressed in terms of the places where information is stored. Example goals would be to *store* or *erase* the value of particular attributes of objects. In Section 5.3.4, the presentation of the use of the hybrid model in blocks-world problems shows that the manner in which goals are expressed does not necessarily impose a limitation on the types of planning problems that may be modeled. In fact, any problem that can be modeled using a means-end model can also be modeled with the hybrid model.

In summary, the hybrid model considers process planning as a procedure in which:

- simple operators successively transform design information into process information; and
- dependencies among these operators are established explicitly in terms of the data dependencies (e.g., the data required by and produced by these operators).

As a result, the hybrid model has two major advantages in modeling process planning problems. First, it overcomes the need for the unified description of the solution space required by the means-end planners because there are operators that transform one description into another. Second, it simplifies the problem of selecting the next operator because control utilizes a declarative representation of operators based on their individual preconditions and effects. Of course, this model does not necessarily eliminate the frame problem of classic planners (see Section 2.1.5) because data items may have unrecorded interrelationships. Nevertheless, explicit recognition of operator data dependencies simplifies this frame problem in many instances.

3.2 Two Illustrative Models for Process Planning Operators

Modeling process planning problems using the hybrid model requires identification of the individual planning operators which create the plan and definition of the manner in which each operator acts. Thus, a model for each process planning operator needs to be developed. Because of the similarities among construction and manufacturing processes, some of these models may be common to several domains. For example, computing scheduling information using a Basic CPM algorithm (see p. 38) is appropriate for any project network with only finish-to-start precedences among the activities. However, another type of model (e.g., the Precedence Diagramming Method) is required when different types of precedences are present.

In general, models for process planning operators can be categorized with respect to the elements of process planning into:

- *Feature Recognition Models* including models used to identify or interpret features of the design;
- *Activity Formulation Models* including models for the identification of work tasks and precedence relationships;
- *Technology Choice Models* including models for the selection of technologies and methods for performing construction or manufacturing activities;
- *Estimation Models* including models used to estimate activity durations and costs based on the technologies and process methods selected for the activities; and
- *Scheduling Models* including models that are used to prepare and maintain activity schedules.

However, some domains may require models whose scope spans multiple categories (e.g., models that combine technology selection and scheduling).

In Chapter 2, models for hierarchical duration estimation, technology selection and project scheduling were presented. The discussion below presents two models which were used in some of the applications of PLANEX to generate and schedule activity networks.

3.2.1 “Bottom-Up” Activity Formulation Model

Several planners generate activity networks using a *top-down* approach in which an aggregate network is successively expanded until an appropriate level of disaggregation is obtained. Although this approach has been successfully used in some domains, the amount of domain knowledge needed for the expansion process is considerable, because many abstraction levels are involved. As an example, suppose that a planner is generating an activity network for constructing a building using the top-down model. An initial network might have aggregate activities such as *Excavate*, *Structural Erection* and *Finishes*, because these activities are required for *all* buildings. However, it is difficult to generate any other information at this aggregate level (e.g., the lag between the excavation and erection activities) without having domain-specific knowledge related to other characteristics of the design.

Figure 3-5 shows an alternate activity formulation model which proceeds in a *bottom-up* fashion. A planner using this model would obtain an activity network by following four steps:

- Step 1. Product Decomposition.* The final product is decomposed in terms of primitive components called *design elements*.
- Step 2. Identify Activities.* The planner determines the activities required to produce each design element. These tasks are denoted as the *element activities* of the design element.

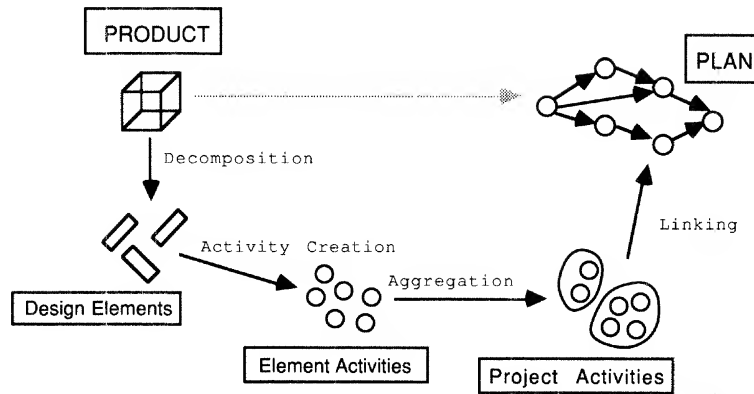


Figure 3-5. *Bottom-Up Activity Formulation Model*

Step 3. Aggregate Activities. The system aggregates element activities into more manageable tasks called *project activities*.

Step 4. Link Activities. The system establishes precedence relationships among project activities in order to create a project activity network.

There are several reasons to favor the use of the bottom-up activity formulation model in process planning problems:

- the bottom-up approach has many similarities with the cost estimating and material requirement processes used by planners;
- activity formulation knowledge is more readily available in the bottom-up approach because cost estimating and process literature usually focus on product or building components; and
- a top-down approach must reach a detail level corresponding to the element activity level in order to perform quantity take-off calculations.

The use of the bottom-up model in construction and manufacturing process planning is illustrated in later chapters.

3.2.2 *Unified Activity Network Model*

The *unified activity network model* represents each project activity by a *start node*, a *finish node* and a connecting *link*. With this model, precedence relationships and activity window constraints are also represented by links, and milestones are modeled as nodes in the activity network. The model is an extension of a similar representation suggested in Bell [5] for use in an automated planning system using AI techniques. The unified activity network model is described more fully in Hendrickson and Au [50] and in Hendrickson and Zozaya-Gorostiza [51]. The major advantage of the unified activity network

model is that it permits the use of standard shortest-path algorithms (e.g., the Basic CPM) in scheduling activities and computing milestone, earliest and latest start and finish times.

The unified model is based on an activity network of nodes and links. Nodes represent *events* (e.g., the start and finish of an activity), including project start and completion. Links are characterized by a minimum duration and a cost. The preceding event time plus the duration must be less than the succeeding event time for each link.

Figure 3-6 illustrates a small unified network model with two activities: i and j . In this model, nodes represent project milestone events (such as the project start [PS], project finish [PF] and activity start and finish [S_i , F_i]). Links represent activities (such as i and j in Figure 3-6), activity precedences or window constraints. Each link k has an associated duration D_k which must be positive or zero. In particular, the eight different precedence and window constraint types illustrated in Figure 3-6 are:

1. The project start must precede the start of activity i by at least D_1 .
2. The project start must precede the finish of activity j by at least D_2 .
3. The start of activity i must precede the start of activity j by at least D_3 .
4. The start of activity i must precede the finish of activity j by at least D_4 .
5. The finish of activity i must precede the start of activity j by at least D_5 .
6. The finish of activity i must precede the finish of activity j by at least D_6 .
7. The finish of activity j must precede the project finish by at least D_7 .
8. The start of activity i must precede the project finish by at least D_8 .

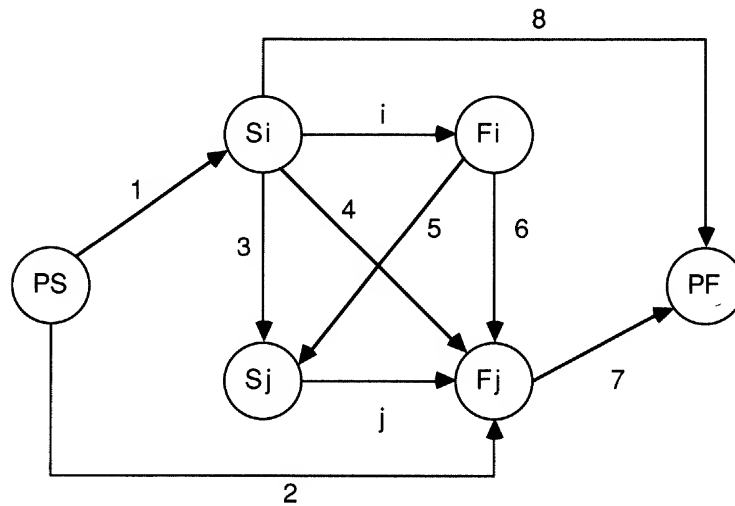


Figure 3-6. Two-Activity Network with Eight Precedence and Window Constraints

These eight links represent four types of precedence relationships and four types of *minimum* or *greater-than* window constraints (numbers 1, 2, 7 and 8).

If negative link durations are permitted, eight additional types of constraints can be represented. A negative link duration imposes a maximum precedence lead. If event k must occur within a prescribed time period $|D_{kh}|$ time units after event h , then a link from k to h with negative duration D_{kh} requires that the time of event k , E_k , is less than or equal to the time of event h plus the prescribed lead $|D_{kh}|$: $E_k + D_{kh} \leq E_h$ or $E_k \leq E_h + |D_{kh}|$ for $D_{kh} \leq 0$. As shown in the partial network of Figure 3-7, links 9 to 16 represent the following constraints:

9. The start of activity i is within $|D_9|$ of the project start.
10. The finish of activity j is within $|D_{10}|$ of the project start.
11. The start of activity j is within $|D_{11}|$ of the start of activity i .
12. The finish of activity j is within $|D_{12}|$ of the start of activity i .
13. The start of activity j is within $|D_{13}|$ of the finish of activity i .
14. The finish of activity j is within $|D_{14}|$ of the finish of activity i .
15. The project finish is within $|D_{15}|$ of the finish of activity j .
16. The project finish is within $|D_{16}|$ of the start of activity i .

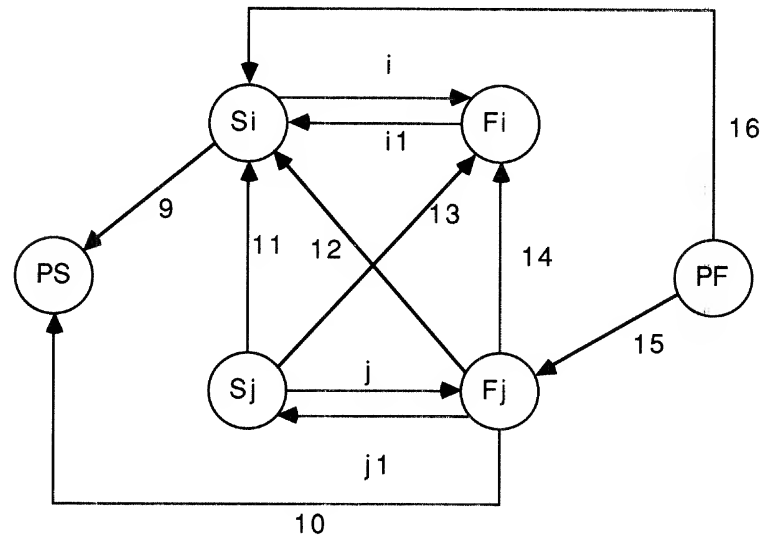


Figure 3-7. Two-Activity Network with Eight Maximum Duration Links

Again, eight different precedence and window constraints exist of a *maximum* or *less-than* type. Unfortunately, negative length cycles may be introduced in the network by permitting negative links, and to avoid cyclic computations, the critical path solution algorithms are more complicated.

Each link duration and event time can also be a variable and can be associated with a cost function. In this case, the network model is the familiar time-cost trade-off model. Another variation is to allow *OR* nodes as in the Decision CPM model (see p. 44).

Figure 3-8 shows an application of the basic unified model to a project with five activities and twelve precedence relationships. In the absence of negative precedence durations, and assuming that all activities are amenable to *splitting*, the unified model can be solved by applying a node labeling algorithm such as the Basic CPM or Dijkstra's [61].

Figure 3-9 summarizes the scheduling results for the small project network of Figure 3-8. In this case, the critical path (illustrated in bold) includes: *PS* → *SC* → *FC* → *SE* → *PF*. The following observations can be made from the solution:

- Both the *start* and *finish* times of activity *C* are critical (i.e., these activities cannot be delayed without delaying the completion time of the project).
- Only the *start* of activity *E* is critical. The *finish* of activity *E* can be delayed three days. Thus, activity *E* can have an overall duration between two and five days (e.g., by splitting its execution) without affecting the completion time of the project.
- All other activity events are non-critical.

Problems of computing and interpreting *floats* or *slacks* in project networks with different types of precedences have been noted by several researchers [73]. In the unified model, node and link floats are computed without modifying the basic solution algorithm. Node floats, which represent the amount of time that an event can be delayed without affecting the total duration of the project, are computed by subtracting the latest event time, $L(i)$, from its corresponding earliest event time, $E(i)$. Link floats for both activities and precedence constraints are computed on the basis of the following definitions [50]:

- *Total Float* is the maximum delay which can be assigned to any one activity or constraint without delaying the entire project. The total float is calculated as $L(j) - E(i) - D_{ij}$ for link (i, j) .
- *Free Float* is the delay which can be assigned to any one activity or constraint without delaying subsequent activities. The quantity $E(j) - E(i) - D_{ij}$ is the free float associated with link (i, j) .
- *Independent Float* is the delay which can be assigned to any one activity without delaying subsequent activities or restricting the scheduling of preceding activities. The independent float for link (i, j) is computed as: $\text{Maximum}(0, E(j) - L(i) - D_{ij})$.

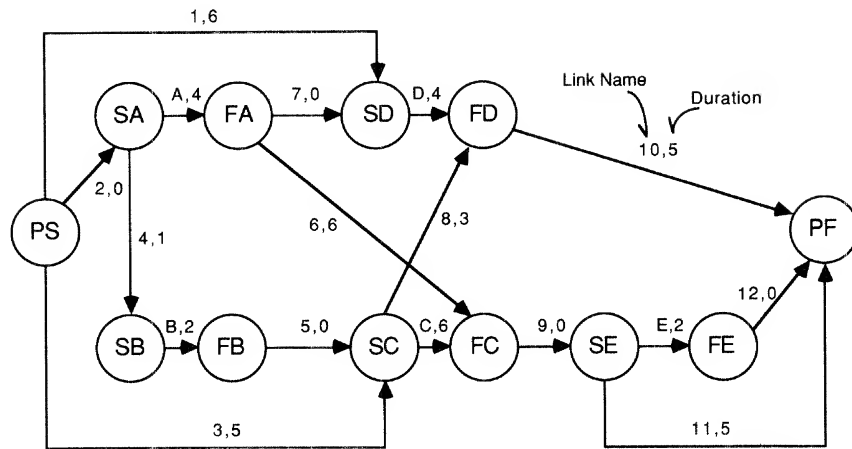


Figure 3-8. Example of the Basic Unified Model

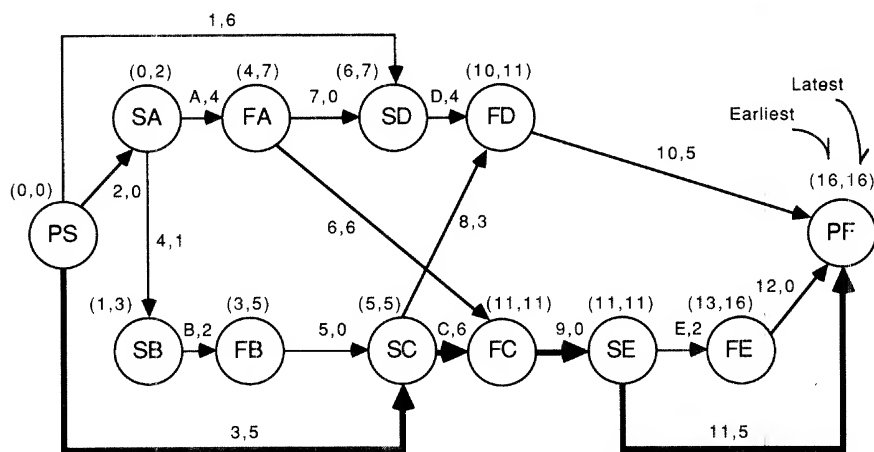


Figure 3-9. Solution to the Basic Unified Model

The unified network model can be used when activity splitting is restricted. Figure 3-10 shows the solution to a modified version of the example of Figure 3-8 for the case in which no activities may be split (i.e., their durations are fixed). In this example, the start time of activity *D* has been constrained to be the same as the finish time of activity *A*. Under this assumption, the total duration of the project is seventeen units, and critical path (shown in bold) is: *PS* → *SD* → *FA* → *FC* → *SE* → *PF*. The earliest-start-time of activity *A* has been set to two units even though there is no window constraint imposed on this event. This is because the finish time of activity *A* is critical and its duration is fixed.

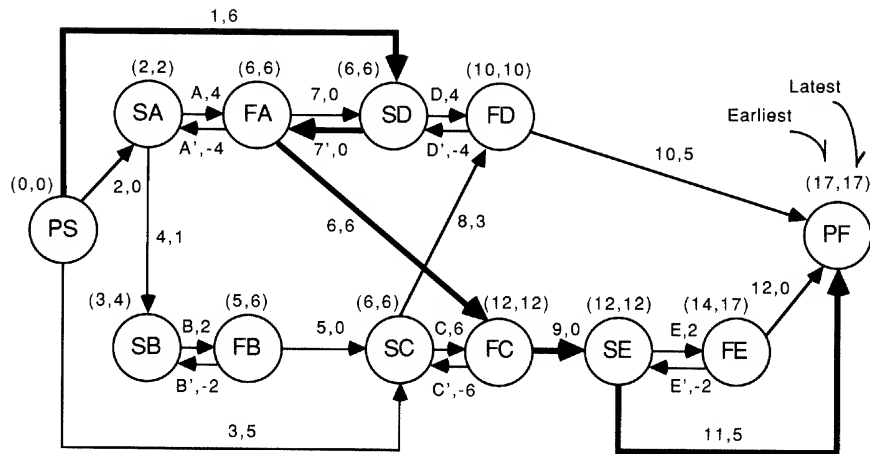


Figure 3-10. Solution to the Unified Model with Maximum Durations

As noted above, allowing negative link durations permits a greater variety of precedence relationships, but also complicates the scheduling computations. Most importantly, a cycle with positive length may appear in the network. In this situation, the Floyd-Warshall algorithm is applicable [61]. This algorithm computes a matrix of shortest paths throughout a network and identifies any positive length cycles. However, the worst-case performance of the Floyd-Warshall algorithm is proportional to the number of nodes cubed ($O(n^3)$).

The development of the unified network model was motivated by the desire to have an activity network model which solved time constraints using simple scheduling procedures. The model accomplishes this objective and provides a means for computing activity floats when various types of precedence relationships are present. However, there are some losses in computing efficiency associated with the use of the model. Since the computational performance of a shortest-path, node-labeling algorithm is proportional to the number of nodes squared ($O(n^2)$) [61], solving a unified network model may take up to four times longer than solving a corresponding conventional network model. When negative links are present, the worst-case performance may be eight times longer since a shortest-path algorithm that checks for positive length cycles is required. Despite these drawbacks, there are some important computational gains in using the unified network model:

- The event, activity and constraint float times are readily available. The activity-on-node (AON) model requires additional computations to produce these values.
- The node-labeling calculations are simpler using the unified network model since no identification of alternative types of links is required. With a

- simpler, general-purpose algorithm, it should be easier to optimize the implementation of a solution algorithm for the unified network model.
- The density of links decreases when using the unified model because the average number of links per node is likely to be smaller than in the equivalent AON model. As a result, the average performance of the solution algorithm is better than indicated by the worst-case analysis.

3.3 Requirements for a Process Planning Architecture

This section specifies requirements for a process planning architecture, PLANEX, that implements the hybrid planning model described above. Specifying these requirements provides a basis for evaluating the structure and behavior of PLANEX, as well as other process planning architectures.

The identification of the requirements followed an iterative process involving both specification of requirements and prototyping. Initially a list of general requirements was developed with respect to the scope of the architecture and its general behavior. After developing a prototype expert system for excavation tasks [49], this initial list was refined and additional architectural requirements were introduced. These requirements led to the development of an expert system for construction planning, called CONSTRUCTION PLANEX, which is described in Chapter 6. The experiences acquired in the process of developing this second prototype led to the more refined list of system requirements presented below. The current PLANEX architecture was implemented to meet these requirements, and the prototypes were reimplemented using this architecture.

3.3.1 General Requirements

In the summary section of Chapter 2, the following reasons for the minimal impact of existing computer tools in solving process planning problems were suggested: (1) tools focus on analysis rather than on synthesis; (2) they do not incorporate domain-specific knowledge; (3) they are not transparent to the user; and (4) they are not flexible. Therefore, an architecture designed to overcome these limitations must be:

- *generic* so it may be used in different domains (e.g., planning the construction of a building, planning the manufacturing operations for a product) without having to modify its internal behavior or structure;
- *transparent* in order to facilitate the understanding of its structure and behavior by its users; and

- *extensible* so that it is flexible enough to extend the scope of a particular application of the architecture.

This section presents a list of proposed requirements for achieving generality, transparency and extensibility. These requirements are classified with respect to four perspectives:

- *Knowledge Representation*, including issues related to how the architecture should store process planning knowledge;
- *Problem-Solving Operators*, involving requirements of how the architecture should use knowledge to perform planning tasks;
- *Control*, concerning behavioral goals detailing how the architecture should apply problem-solving operators to obtain a process plan; and
- *User Interaction*, involving issues related to facilities available to the user to override decisions, obtain information, modify system knowledge or change the set of problem-solving operators.

The first three perspectives refer to intrinsic characteristics of the architecture that affect how the system manipulates knowledge to solve problems. The fourth perspective involves exogenous elements used to modify the behavior of the architecture or to adapt it to other process planning domains. The four perspectives are not independent. For example, modifying the knowledge of a system is a user interaction issue directly related to the type of knowledge representation employed. Similarly, the characteristics of the knowledge representation affect the problem-solving behavior of the system.

3.3.2 *Knowledge Representation*

A fundamental issue in a knowledge-based architecture such as PLANEX is the form in which knowledge is represented. With respect to this issue, the following requirements have been identified. PLANEX should:

1. *Provide a process-independent knowledge representation.* PLANEX should provide a means of encoding knowledge for different process planning domains using a uniform, domain-independent format. This would facilitate adapting PLANEX for different application domains.
2. *Provide an operator-independent knowledge representation.* PLANEX should provide a means for encoding knowledge about different process planning tasks (i.e., operators) such as activity identification or duration estimation in a uniform, operator-independent format. This would facilitate the extensibility and transferability of expert knowledge.
3. *Provide the means to structure knowledge hierarchically.* PLANEX should provide a means to structure the large body of knowledge needed in process planning. Experiences with the prototypes for construction process planning

suggest that a good strategy is to structure this knowledge into hierarchies. Hierarchies may be established on the basis of the type of operators using this knowledge (e.g., duration estimation versus equipment selection) or on the basis of the level of detail represented in the knowledge (e.g., selecting a general type of excavator versus selecting a type of power shovel).

4. *Provide the means to check the completeness or consistency of an operator's knowledge.* For a particular planning operation, the user may want to check whether all possible combinations of factors affecting the outcome have been considered in formalizing the domain knowledge. In other cases, the user may require that the knowledge associated with a particular operator provides a unique result for any combination of factors. The knowledge representation of PLANEX should be amenable to supporting such checking.

3.3.3 Problem-Solving Operators

All computer systems contain a set of problem-solving operators that define the actions executed by the system when solving a problem. In some programs, the set of operators is fixed and the program's structure is not transparent to the user. Most commercial scheduling packages are examples of such programs. The user's only interaction with these systems takes place during the formulation of the input and the interpretation of results. In solving a problem, these systems perform as a *black-box* whose structure is fixed. A more flexible approach is desirable. The following requirements are related to the PLANEX operators. PLANEX should:

1. *Achieve operator modularity.* It should be possible to decompose the set of operators into elementary operators whose scope and purpose are easy to identify. This characteristic would improve the flexibility of the architecture.
2. *Provide a set of problem-solving operators that may be used in different process planning domains.* There are several operations that are common to any process planning domain. For example, the Basic Critical Path Method may be used to compute the duration of a project regardless of the nature of the activities in the project network. PLANEX should provide a set of common operators in order to facilitate the development of application programs.
3. *Incorporate both synthesis and analysis operators.* As previously discussed, one of the problems with computer tools for process planning is that they only perform analysis tasks and leave synthesis tasks to the planner. In an *integrated* system, it is important to perform both of these tasks. For example, synthesis tasks are required when generating project activities and selecting appropriate construction and manufacturing methods. Analysis tasks are required to determine the consequences of planning actions. (Examples of analysis operators are those that estimate the duration and cost of project activities.) Both types of operators should be supported.

4. *Provide the means to structure operators hierarchically.* In some domains, it may be useful to structure operators into hierarchies. Hierarchies may be established on the basis of the level of generality of the operator (estimating the cost of the project versus estimating the cost of concrete columns) or on the basis of the purpose of the operator (estimating costs versus estimating durations). As described in Section 5.1.2, operator hierarchies are used to perform hierarchical planning in a similar manner to that of ABSTRIPS. The architecture should support such hierarchical structuring of operators.

3.3.4 Control

In addition to identifying requirements with respect to operator characteristics, another important issue to consider is that of *control*. Control defines the behavior of a system in terms of selecting and executing operators during the planning process. In defining a control architecture (see p. 32), Hayes-Roth proposed the following behavioral goals for an intelligent system [45, p. 252]:

- (1.) Make explicit control decisions that solve the control problem.
- (2.) Decide what actions to perform by reconciling independent decisions about what actions are desirable and what actions are feasible.
- (3.) Adopt variable grain-size control heuristics.
- (4.) Adopt control heuristics that focus on whatever action attributes are useful in the current problem-solving situation.
- (5.) Adopt, retain, and discard individual control heuristics in response to dynamic problem-solving situations.
- (6.) Decide how to integrate multiple control heuristics of varying importance.
- (7.) Dynamically plan strategic sequences of actions.
- (8.) Reason about the relative priorities of domain and control actions.

Although these are desirable behavioral goals for an intelligent planner that decides which operators to execute during process planning, a distinction should be made with respect to the plausibility of achieving these goals in a *generic* process planning system. On the one hand, some of the goals seem easily achievable in any process planning domain. For example, any system that does not execute operators in a predetermined manner would make decisions about which operators to apply at any point in the planning process. Even if these decisions are made solely on the basis of the applicability of the operators, the system satisfies the first behavioral goal. On the other hand, other goals would only be satisfied if the system is provided with considerable control knowledge. Incorporating these goals into system requirements would suggest an architecture that encodes control knowledge in structures similar to the *Choose-KSAR* and *Refine-or-Chain?* knowledge sources of OPM. For example, to dynamically

use various control heuristics (the fifth item in Hayes-Roth's list, above) the system needs to possess knowledge about the applicability and desirability of alternative control heuristics at different stages of the planning process. It is an open question whether such knowledge is dependent on the type of domain being modeled. The fact that a general theory of process planning has not been developed may indicate that this type of knowledge is domain-specific. If this is the case, satisfying all of Hayes-Roth's behavioral goals in the architecture of PLANEX would make adapting it to different domains complex. For each domain, the user would have to provide the system not only with the *domain* knowledge (e.g., possible activities, appropriate resources, productivities, costs) but also with *control* knowledge (e.g., in planning building construction, choosing the general construction method before defining the activities).

The following are the basic control requirements proposed for the architecture. PLANEX should:

1. *Make explicit control decisions that solve the problem.* Planning is not a straightforward process. There are many interactions that must be considered because one decision often affects others. A good sequence of operations for a certain project (e.g., determine the general type of excavation equipment first) may not be the proper sequence for other projects. Because the sequence of planning operations is not fixed, PLANEX should make control decisions regarding the order in which different operators are executed.
2. *Decide what operators to execute in terms of their feasibility and desirability.* At any point in the solution process there may be several applicable operators. However, only some of them may contribute to the solution of the problem. The distinction between feasibility and desirability has been incorporated into planning systems since the development of GPS (see p. 16). PLANEX should be able to identify both feasible and desirable operators and decide which one to execute next.
3. *Dynamically plan strategic sets of operators.* During the planning process, the user or the system may establish a goal that is not directly achievable by any feasible operator. The system should be able to elaborate a plan of operators to accomplish the goal.
4. *Incorporate different control heuristics in the planning process.* Accomplishing the other behavioral goals requires different control heuristics be present in the system. For example, for strategic planning of operators, a backward search similar to the goal expansion process of NOAH (see p. 23) may be appropriate. In other situations, a forward search may be required to propagate the consequences of particular operations. PLANEX should provide the mechanism to incorporate such different control heuristics.

Although these requirements do not include all of Hayes-Roth's goals, it seems plausible to implement them in a generic knowledge-based architecture for plan-

ning. This does not mean that the basic system architecture could not be extended to incorporate some of the goals that have been excluded. In particular domains, control knowledge and operators could be added to the basic architecture described in this chapter to yield a more intelligent control behavior.

3.3.5 User Interaction

In a knowledge-based system, it is important to “keep a human in the loop” [107, p. 13]. This is particularly true for a generic system architecture such as PLANEX. User interaction is required to modify the structure, knowledge or behavior of the system in areas such as: (1) adapting and extending the system from one domain to another; (2) debugging the knowledge for a particular application; (3) extending the scope of the problem-solving process in a certain domain; and (4) modifying the manner in which a solution is obtained. The proposed user interaction requirements for PLANEX are:

1. *Provide the means to create, discard or update domain knowledge.* In any process planning domain, significant knowledge is needed. PLANEX should include tools which permit easy modification of the knowledge required for particular domains.
2. *Provide the means to modify the set of operators.* Different domains may require changes to the basic set of operators used by the application system. For example, when planning building construction, some contractors may require that the system determine when the construction materials should be purchased. In this case, the contractor may have to add one or more operators to the planning system to perform this task. Capabilities to modify the set of operators should be included in the system.
3. *Provide the means to control the planning process.* PLANEX is intended to be an assistant that provides the user with several mechanisms to control the planning process. The user should be able to execute individual operators, establish goals or introduce changes that affect the manner in which problem-solving operators are executed.
4. *Explain results in terms of the knowledge used to obtain them.* PLANEX should provide the user with explanations that facilitate the understanding of its problem-solving behavior. This information is needed to expand and refine the knowledge and operators of application systems developed using the architecture.
5. *Provide the means to produce reports with flexible formats.* The architecture should provide the user with the means for designing various types of reports. This capability improves the adaptability of the architecture to different domains.
6. *Provide graphical display of results.* Graphical output is very useful for displaying information. Bar-charts and network diagrams are commonly

used by managers to represent process plans [50, 72]. The architecture should provide the means to incorporate these and other types of graphical output.

3.4 Conclusions

This chapter introduced a hybrid model of process planning suitable for computer implementation. Two illustrative models to perform specific tasks within the planning process were also described. The requirements that a general purpose, knowledge-based process planning system should fulfill were formalized. These models and requirements provide a framework for the development of the PLANEX system architecture.

4 A Knowledge-Based Architecture for Process Planning

This chapter describes a software architecture for process planning, PLANEX, which is designed to satisfy the list of structural and behavioral requirements presented in the previous chapter. PLANEX provides tools for representing and using the knowledge required by process planning operations. These tools have been used to develop process planning systems including CONSTRUCTION PLANEX, a knowledge-based expert system for construction planning which is described in Chapter 6 and HARNESS PLANEX, an expert system for electrical wire harness process planning which is described in Chapter 7.

As noted in Section 1.5, the development of PLANEX followed an iterative prototype building and refinement process. The requirements developed during this process (see Section 3.3) resulted in the flexible system architecture described in this chapter.

This chapter first presents an overview of the basic components of PLANEX and their use and interrelationships in the solution of process planning problems. Subsequent sections describe the PLANEX architecture in terms of three perspectives:

- *Knowledge Representation*—Details of the tools provided by PLANEX to store and retrieve process planning knowledge;
- *Problem Solving and Control*—Discussion of the tools used to implement the behavior of the hybrid process planning model described in Section 3.1.4; and
- *User Interaction*—Explanation of the mechanisms provided to the planner to change decisions, obtain information, modify system knowledge or change the set of planning operators.

4.1 Overview of PLANEX

4.1.1 Basic Components of PLANEX

The knowledge-based system architecture for process planning contains four major components:

1. *Representational Structures.* Information about solution elements, planning decisions and input data for the planning process is stored in a global data store called the *context*. The context contains hierarchical representational structures that are composed of *objects* linked with each other using different types of *relations*. Each object has a unique name and contains *slots* that store *values* of the attributes of the object or pointers to other objects. An object may belong to more than one hierarchical structure. In different applications of PLANEX, there may be schemas for storing information about process components, resources, activities, etc. However, the set of hierarchies is specific to each problem domain.
2. *Operators.* Solution of the process planning problem is achieved by applying problem-solving operators. Each operator is a procedural function that modifies the context by creating, modifying or deleting objects. There are two types of operators: *domain* and *control*. Domain operators perform specific planning tasks such as choosing technologies, estimating activity durations, or scheduling. Domain operators are specific to each problem domain. However, several application systems may share common domain operators. Control operators determine the sequence in which domain operators are executed. The set of control operators is common to all problem domains.
3. *Knowledge Sources.* Knowledge required by the operators is stored in groups of rules called *Knowledge Sources* (KSs). Each KS provides information for individual operators such as activity formulation, duration estimation or precedence determination. Knowledge sources are used to return values to the operators, to modify the context and to invoke the evaluation of other KSs. The *knowledge base* of PLANEX is composed of many KSs.
4. *User Interface.* PLANEX is an interactive system architecture that incorporates a user interface used for modifying any of the three components described above, controlling the execution of the system, changing planning decisions or retrieving information from the context. The user interface is composed of several editors, command menus, graphical displays (both interactive and passive) and a report generator.

4.1.2 Relationships Among the Basic Components of PLANEX

Figure 4-1 illustrates the relationships among an operator, its associated knowledge sources and the different types of context objects related to the operator. Each operator may be applied to different sets of objects called the *application objects* of the operator. For example, if the operator that estimates activity durations is applied to all the activities used to construct a particular floor, these activities constitute the application objects of the operator at that point during the solution of the process planning problem. Usually operators are sequentially applied to their application objects. The object to which the operator is applied is called the *current object*. When an operator is executed, a set of knowledge sources may be evaluated and the results of this evaluation are returned to the operator. With respect to each operator, the context is divided into three different sets of objects: the *input objects*, the *output objects* and the *remaining objects* (those not associated with the operator). The set of input objects of the operator contains all the data required to evaluate the related knowledge sources and to perform procedural calculations. The objects affected by the application of an operator and the evaluation of its associated knowledge sources constitute the set of output objects of the operator. The sets of input and output objects may have some common elements. Usually the current object belongs to both sets.

When a domain operator is executed, either in response to a user request or by a control operator, the following steps are performed:

- Step 1. Identify the application objects.* The objects to which the operator is applied are the application objects.
- Step 2. Select a KS to be used by the operator.* Each type of operator (e.g., technology choice, duration estimation) is related to a specific set of KSs. These KSs contain all of the knowledge required to perform the specific operation. In some cases, knowledge sources are not required for operators performing procedural calculations.
- Step 3. Select one specific object from the application objects.* This object becomes the *current object* used when evaluating the KS. Evaluating a KS may or may not require information from the selected object.
- Step 4. Evaluate the selected KS.* The selected KS is evaluated using the KNOWLEDGE SOURCE EVALUATOR (KSE) and the results are returned to the domain operator. The domain operator is responsible for processing these values, either storing them in schemas or using them to perform other computations.

Some of these steps may be repeated depending upon the results of the KS evaluation (e.g., another KS must be evaluated) or the range of the operator (e.g., apply the operator to several application objects).

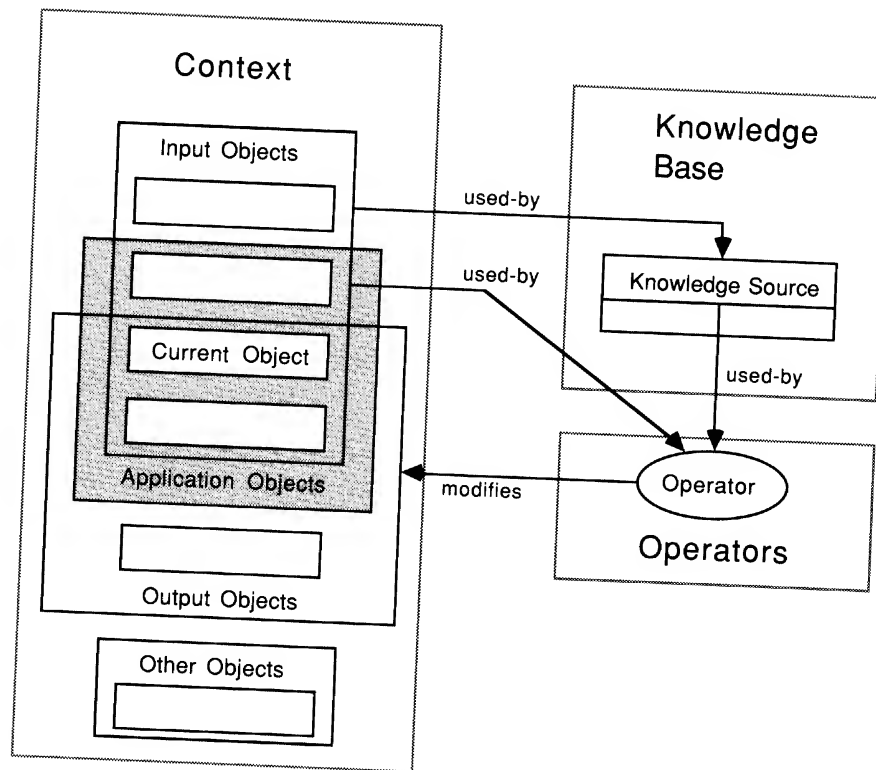


Figure 4-1. Relationships of Operators, Knowledge Sources and Context Objects

The set of context changes caused by a domain operator is known as the *effects* of the operator. Effects are described as changes to slot values of particular objects. An effect is *predictable* if it can be described *before* the operator is executed, and it is *unpredictable* otherwise. The information about the predictable and unpredictable effects of domain operators is stored in context objects containing declarative knowledge which describes the operators. This knowledge is used by the control operators to determine which planning tasks should be performed during the planning process. Operator schemas also contain the names of the KSs used by the operator.

4.1.3 User Interaction Mechanisms

The PLANEX architecture incorporates several user interaction mechanisms:

1. an interactive environment for modifying the knowledge base of the system, called the KNOWLEDGE SOURCE ACQUISITION MODULE (KSAM);

2. an interactive environment for modifying control information called the CONTROL PANEL (CP);
3. a *menu-driven interface* used to control the execution of the domain and control operators used to request explanations of planning decisions;
4. the REPORT GENERATOR (RG) for outputting process information in tabular forms; and
5. several *graphical displays*, some of which are interactive.

In addition, the user may directly modify context objects using the primitive functions of the frame representation language used to implement PLANEX. These functions (e.g., the function to create a new schema) are used to implement the procedural codes of the operators.

4.2 Knowledge Representation

Knowledge-based expert systems contain considerable domain knowledge that must be represented in an organized manner. PLANEX provides means to incorporate and use this knowledge effectively in various application domains. In the development of this architecture, several alternative knowledge representations were explored. The first prototype system for excavation planning used a mixed representation of domain knowledge: isolated rules and procedural functions. The experiences with this prototype showed that this representation was inadequate for acquiring, representing or updating process planning knowledge.

In the prototype for planning the construction of buildings, a knowledge representation scheme based on decision tables was employed. This representation proved to be very effective for acquiring and representing the KSs of PLANEX. Domain knowledge is decomposed with respect to the different types of planning operations associated with the knowledge. Knowledge is represented by one or more KSs. Each KS is a decision table that groups rules which share common antecedents or consequents. The structure of a typical decision, the mechanism of KS implementation and interpretation and some capabilities and limitations of the knowledge representation are discussed below.

4.2.1 Structure of a Decision Table

Research on the decision table as a symbolic mechanism to represent logical interdependencies among events has been underway for over three decades. Reviews of the basic theory of decision tables and their applications in software engineering is given by Hurley [53] and by Welland [108]. Advantages include the compactness and transparency with which knowledge is expressed and the possibility of checking for omissions and logical inconsistencies in the decision tables. In civil engineering, the use of decision tables to represent construction specifications and standards was initially proposed more than twenty years

ago [31]. Recently some expert systems have used decision tables to represent domain knowledge. Examples include the SPEX [38] and TRALI [114] systems. SPEX, an expert system for designing structural components of a building, uses decision tables to represent design standards. TRALI, an expert system assistant for traffic signal setting, uses decision tables to find crossing conflicts between pairs of traffic flows.

Figure 4-2 shows a simplified decision table used for determining the activities required to construct a cast-in-place concrete column footing. This table returns a list of activity names to the operator responsible for activity creation. The table contains three regions:

- *conditions* represented as boolean predicates in the left part of the upper rows of the table;
- *actions* represented by values or functions in the left part of the bottom rows of the table; and
- *rules* represented in the columns of the right side of the table. Each column specifies the conditions that should be True ("T"), False ("F") or Irrelevant ("I"), and those values or actions that will be executed ("X") or Ignored ("I") for that rule (see p. 98 for an explanation of how irrelevant values are handled when evaluating the decision table).

KS-Example			
Conditions	Rules		
design element is a cast-in-place concrete column-footing	T	T	F
soil is appropriate for backfill	T	F	I
Actions			
excavate-column-footing	X	X	I
dispose-of-excavation-column-footing	I	X	I
pile-up-excavation-column-footing	X	I	I
borrow-material-column-footing	I	X	I
place-forms-column-footing	X	X	I
reinforce-column-footing	X	X	I
pour-concrete-column-footing	X	X	I
remove-forms-column-footing	X	X	I
KS-other-elements	I	I	X

Figure 4-2. Example of a Decision Table

In this example, knowledge about construction activities is represented by three rules:

- Rule 1.* If the component is a cast-in-place concrete column footing and if the soil can be used for backfill, the required activities are: (1) excavate for the footing; (2) pile up the excavation material, (3) place the forms; (4) place the reinforcement; (5) pour the concrete; and (6) remove the forms.
- Rule 2.* If the component is a cast-in-place concrete column footing and if the soil cannot be used for backfill, the required activities are: (1) excavate for the footing; (2) dispose of the excavation material; (3) borrow backfill material; (4) place the forms; (5) place the reinforcement; (6) pour the concrete; and (7) remove the forms.
- Rule 3.* If the component is not a cast-in-place concrete column footing, a different decision table, called *KS-other-elements*, must be evaluated. PLANEX identifies that *KS-other-elements* is a KS using auxiliary information, as described below.

This example is used below to illustrate how decision tables represent knowledge.

4.2.2 Implementation of a Knowledge Source

The knowledge base of PLANEX is composed of many KSs whose structure resembles that of a decision table. A KS contains additional information detailing where and how to retrieve the information required to evaluate the KS. Figure 4-3 shows the KS corresponding to the decision table of Figure 4-2. Each condition is defined by four basic elements:

- *object*, which may be: (1) the name of a context object; (2) "Function"; (3) "KS"; or (4) "None". These values indicate that the condition to be evaluated is: (1) a slot name; (2) a function name; (3) another KS; or (4) the name of a global variable. In the example, both conditions are evaluated using values of context objects.
- *slot*, which may be the name of a slot, function, KS or global variable, depending on the value of the *object* item. In the example, the second condition is evaluated using the value of the slot *possible-use* of the object *soil-characteristics*.
- *op*, which contains an operator such as "=", ">", or "is" that expresses the relationship between the *slot's* value and the *value*.
- *value*, which stores a number or a symbol.

KSs are implemented as context schemas using the syntax defined by the BNF grammar shown in Figure 4-4. This form is used to define all knowledge source

Name: KS-Example		Type: first				
Object	Slot	Op	Value	Rules		
current-object	type-element	is	cast-in-place concrete column-footing	T	T	F
soil-characteristics	possible-use	is	backfill	T	F	I
excavate-column-footing				X	X	I
dispose-of-excavation-column-footing				I	X	I
pile-up-excavation-column-footing				X	I	I
borrow-material-column-footing				I	X	I
place-forms-column-footing				X	X	I
reinforce-column-footing				X	X	I
pour-concrete-column-footing				X	X	I
remove-forms-column-footing				X	X	I
KS-other-elements				I	I	X

Figure 4-3. Example of a Knowledge Source

schemas in PLANEX. The terminal symbol `defschema` is the name of the macro used to create a knowledge source schema. Utilities to convert from the decision table form of Figure 4-3 to schemas exist in the KNOWLEDGE SOURCE ACQUISITION MODULE (see Section 4.4.1). Knowledge sources are represented as schemas so that they can be interpreted by the KNOWLEDGE SOURCE EVALUATOR of PLANEX.

Figure 4-5 shows the schema representation of the example KS. The schema contains the following slots:

- *is-a* identifies the schema as a KS;
- *ks-type* indicates the type of firing mechanism used when evaluating the KS;
- *cond-objects* indicates the source of the information required to evaluate each condition (a context object or another source, such as a function, other KS or global variable);
- *conditions* consists of a list of predicates;
- *lhs-rules* stores the antecedents of the rules;
- *rhs-rules* stores the consequents of the rules; and
- *actions* consists of a list of possible results (functions or KS names).

The evaluation of a KS and how the information in the *actions* slot is interpreted

```

<decision-table>      ::= (defschema <sep> <symbol>
                           (is-a      <sep> KS)
                           (ks-type   <sep> <ks-type-value>)
                           (cond-objects <sep> <cond-objects-list>)
                           (conditions <sep> <conditions-list>)
                           (lhs-rules  <sep> <lhs-rules-list>)
                           (rhs-rules  <sep> <rhs-rules-list>)
                           (actions    <sep> <actions-list>))

<ks-type-value>       ::= first | all
<cond-objects-list>   ::= ( <cond-objects-values> )
<conditions-list>     ::= ( <conditions-values> )
<lhs-rules-list>      ::= ( <lhs-rules-values> )
<rhs-rules-list>      ::= ( <rhs-rules-values> )
<actions-list>        ::= ( <actions-values> )
<cond-objects-values> ::= <object> | <sep> <cond-objects-values>
<conditions-values>   ::= <condition> | <sep> <conditions-values>
<lhs-rules-values>    ::= <lhs-list> | <sep> <lhs-rules-values>
<rhs-rules-values>    ::= <rhs-list> | <sep> <rhs-rules-values>
<actions-values>      ::= <action> | <sep> <actions-values>
<object>              ::= <symbol> | KS | Function | None
<condition>           ::= ( <slot-fcn-ks> <sep>
                           <bool-operator> <sep>
                           <symbol-binding-value> )

<lhs-list>            ::= ( <tfi-values> )
<rhs-list>            ::= ( <xi-values> )
<action>              ::= <symbol-binding> | <number> |
                           <fcn-args> | <quoted-list>

<slot-fcn-ks>         ::= <symbol-binding> | <fcn-args>
<bool-operator>       ::= is | = | < | <= | > | >= | <> | member |
                           not-member
<symbol-binding-value> ::= <symbol-binding> | number
<tfi-values>          ::= T | F | I | <sep> <tfi-values>
<xi-values>           ::= X | I | <sep> <xi-values>
<fcn-args>            ::= ( <symbol-binding-list> )
<quoted-list>         ::= ' ( <symbol-binding-list> )
<symbol-binding-list> ::= <symbol-binding> | <sep> <fcn-args> |
                           <sep> <quoted-list> |
                           <sep> <symbol-binding-list>

<symbol-binding>      ::= <symbol> | <binding> |
                           <symbol> <symbol-binding> |
                           <binding> <symbol-binding>

<sep>                 ::= <separator> | <separator> <sep>
<number>              ::= any number type in COMMON LISP (e.g.,
                           fixed, float, rational)
<separator>           ::= a space, line-feed, or new-line
<symbol>              ::= an identifier which may include:
                           + - / $ % & _ = ~ .
<binding>             ::= a symbol enclosed in < >

```

Figure 4-4. Knowledge Source Schema Definition Grammar

```

(defschema ks-example
;
; This ks identifies the element activities required to build a
; cast-in-place concrete column footing
;
  (is-a      ks)
  (ks-type   first)
  (cond-objects current-object soil-characteristics)
  (conditions (= type-element
                    cast-in-place-concrete-column-footing)
              (= possible-use backfill))
  (lhs-rules (T T)
             (T F)
             (F I))
  (rhs-rules (X I X I X X X I)
             (X X I X X X X I)
             (I I I I I I I X))
  (actions  excavate-column-footing
             dispose-of-excavation-column-footing
             pile-up-excavation-column-footing
             borrow-material-column-footing
             place-forms-column-footing
             reinforce-column-footing
             pour-concrete-column-footing
             remove-forms-column-footing
             ks-other-elements))

```

Figure 4-5. Schema Representation of the Knowledge Source of Figure 4-3

(e.g., whether the values are symbols, functions or other KSs) is discussed in the next section.

4.2.3 Knowledge Source Evaluation

Knowledge sources are evaluated by a generic operator which is called the KNOWLEDGE SOURCE EVALUATOR. When an operator requires the evaluation of a particular KS, it invokes the KSE, supplying the name of the KS to be evaluated and the name of the domain object to which the operator is applied. The KSE evaluates the KS and returns a list of result values to the operator. The operator acts upon the list (e.g., performs arithmetic operations or modifies context objects).

The algorithm used by the KSE to evaluate a KS is detailed below. The following lists are used in the algorithm to store values of the slots of the KS being evaluated:

- *objects-list*, O, stores the values of the *cond-objects* slot. The elements of this list are symbols.

- *conditions-list*, X , whose elements are the values of the *conditions* slot. Each list element is a triplet of the form $(operator\ e_2\ e_3)$. The KSE determines if e_2 is a function, a symbol or the name of another KS using the value of the corresponding element of the *objects-list*.
- *actions-list*, A , stores the values of the *actions* slot. Each element is either a number, a symbol or a function. If the element is a symbol, the KSE searches the *is-a+inv* pointer of the *KS* schema to determine if the symbol is the name of a KS.
- *lhs-list*, Λ , whose elements are the values of the *lhs-rules* slot. Each element is a list of "T", "F" or "I" terms. The number of terms in each list is equal to the number of conditions in the KS.
- *rhs-list*, P , whose elements are the values of the *rhs-rules* slot. Each element is a list of "X" or "I" terms. The number of terms in each list is equal to the number of actions in the KS.

In addition, the algorithm uses four other lists:

- *bindings-list*, whose elements are of the form $(binding\ value)$. The first term of a pair is a *binding*⁶ such as $\langle x \rangle$ or $\langle floor \rangle$ and the second term is the value associated with this binding.
- *matchings-list*, whose elements are lists composed of "T", "F" or "B" terms, where "B" stands for binding. The length of the list is equal to the number of conditions in the KS.
- *results-list* stores the list of results before they are interpreted by the KSE.
- *final-results-list* stores the list of results after interpretation by the KSE. These values are returned to the operator which invoked the KS evaluation.

The algorithm proceeds as follows. All conditions are evaluated and classified as being true ("T"), false ("F") or a binding condition ("B"). This information is stored in the *matchings-list*. The left-hand-side of each rule is compared to this list to determine if the rule is to be fired. Depending on the type of *firing mechanism* used to evaluate the KS, rules may be fired sequentially or a more elaborate scheduler may be used to select one of the rules whose antecedent satisfies the *matchings-list*. In the algorithm presented below, firing is sequential. Each time a rule is fired, the actions indicated in its consequent or right-hand-side are executed and the resulting values or actions are stored in the *results-list* for later interpretation. After all of the rules have been fired, the values of *results-list* are interpreted and the *final-results-list* is returned to end the evaluation process.

⁶ Bindings are always enclosed in angle brackets ($\langle \rangle$). The binding is the name of a symbol which is evaluated at run-time to obtain an actual value (i.e., the *binding value*). This value is substituted for all occurrences of the binding term in the KS. Bindings are used in a similar fashion in all of the representational structures of PLANEX.

The algorithm uses auxiliary procedures to substitute binding variables and to select the rules:

- *Bind Variables.* This procedure searches for binding variables in the *conditions-list* or *actions-list* and substitutes the corresponding values from the *bindings-list*. For example, if the current value of the *bindings-list* is:
 $((\langle x \rangle \text{ pour-concrete}) (\langle \text{floor} \rangle 5))$
 and the value of the actions list is:
 $(\langle x \rangle \text{-columns-}\langle \text{floor} \rangle)$
 this procedure performs two substitutions and the new value becomes:
 $(\text{pour-concrete-columns-}5)$
- *Match Conditions with Rules.* This procedure compares the complete *matchings-list* with an element of the *lhs-list*. In this comparison, any binding (indicated with "B") or any *lhs-list* value equal to "I" are ignored. For example, if the *matchings-list* is "(T B F)", evaluation of *lhs-list* elements "(T F F)" or "(T T I)" would return "T", while evaluation of "(T F T)" or "(F T I)" would return "F". This treatment of irrelevant ("I") provides some flexibility in knowledge representation. For example, when a set of conditions is used to determine if a value is one element of an exclusive set, irrelevant tests are employed instead of false tests (e.g., to test if A is "X" or "Y", rules "((T I) (I T))" are used in place of "((T F) (F T))"). Also, rules of the form "(I I ... I)" represent an *all* rule which fires for any set of conditions. The flexibility implies that the rules need not be consistent or complete.

Algorithm for the KNOWLEDGE SOURCE EVALUATOR

Step 1. Initialize

- 1.1 Let KS^* be the name of the KS being evaluated.

Step 2. Evaluate Conditions

2.1 Initialize Lists.

Assign *ks-type* the value of the *ks-type* slot of schema KS^* .
 Assign *O* the value of the *cond-objects* slot of schema KS^* .
 Assign *X* the value of the *conditions* slot of schema KS^* .
 Assign *A* the value of the *actions* slot of schema KS^* .
 Create empty lists for the *matchings-list* and *bindings-list*.

2.2 Loop through Conditions.

If *X* is empty: go to Step 2.11.
 Let *o* and χ be the first elements of *O* and *X* respectively.
 Let $O \leftarrow O \setminus \{o\}$.
 Let $X \leftarrow X \setminus \{\chi\}$.

- 2.3 Use procedure *Bind Variables* to substitute binding values in condition χ .

- 2.4 If o indicates "KS" evaluation: go to Step 2.5.
 If o indicates "Function" evaluation: go to Step 2.6.
 If o is "none": go to Step 2.8.
 Go to Step 2.7.
- 2.5 *KS Evaluation of a Condition.*
 Let KS^{**} be the first element of condition χ .
 Evaluate KS^{**} and replace KS^{**} by the first value resulting from this evaluation in condition χ .
 Go to Step 2.9.
- 2.6 *Function Evaluation of a Condition.*
 Let F^{*} be first element of condition χ .
 Evaluate F^{*} and replace F^{*} by the result of evaluating the function in condition χ .
 Go to Step 2.9.
- 2.7 *Schema Evaluation of a Condition.*
 Let $slot^{*}$ be the value of the first element of condition χ .
 Replace $slot^{*}$ by the value of slot $slot^{*}$ of schema o in condition χ .
- 2.8 *Binding Evaluation.*
 If the third element of condition χ is a binding variable: go to Step 2.9.
 Let m be the result of evaluating condition χ (m is "T" or "F").
 Go to Step 2.10.
- 2.9 *Augment Bindings-list.*
 Let e_2 and e_3 be the second and third elements of predicate χ .
 Let $bindings-list \leftarrow bindings-list \cup \{ (e_3 \ e_2) \}$.
 Let $m \leftarrow \{b\}$.
- 2.10 *Augment Matchings-list.*
 Let $matchings-list \leftarrow matchings-list \cup \{m\}$.
 Go to Step 2.2.
- 2.11 *Translate Actions.*
 Use procedure *Bind Variables* to substitute binding values in the list of actions A .

Step 3. Select and Execute Rules

- 3.1 *Initialize Lists.*
 Assign Λ the value of the *lhs-rules* slot of schema KS^{*} .
 Assign P the value of the *rhs-rules* slot of schema KS^{*} .
 Create an empty list for *results-list*.

3.2 *Loop through Rules.*

If either Λ or P is empty: go to Step 4.

Let λ and ρ be the first elements of Λ and P respectively.

Let $\Lambda \leftarrow \Lambda \setminus \{\lambda\}$.

Let $P \leftarrow P \setminus \{\rho\}$.

3.3 If λ matches *matchings-list* (using procedure *Match Conditions with Rules*): go to Step 3.4.

Go to Step 3.2.

3.4 Find actions B of A corresponding to “X” values in ρ .3.5 Let *results-list* \leftarrow *results-list* \cup $\{B\}$.3.6 If *ks-type* is “all”: go to Step 3.2.

Go to Step 4.

*Step 4. Interpret Actions*4.1 *Initialize List of Results.*

Create an empty list for *final-results-list*.

4.2 *Loop through Results.*

If *results-list* is empty: stop and return *final-results-list*.

Let α be the first element of *results-list*.

Let *results-list* \leftarrow *results-list* $\setminus \{\alpha\}$.

4.3 If α is a number or a symbol not in the *is-a+inv* slot of schema *KS*: let *final-results-list* \leftarrow *final-results-list* \cup $\{\alpha\}$; go to Step 4.2.4.4 *Recursion in Actions.*

If α is a symbol included in the *is-a+inv* slot of schema *KS*: let β be the result of evaluating *KS* α ; let *final-results-list* \leftarrow *final-results-list* \cup $\{\beta\}$; go to Step 4.2.

4.5 *Function Evaluation.*

If α is a list whose first element is the name of a function: let β be the result of evaluating α ; let *final-results-list* \leftarrow *final-results-list* \cup $\{\beta\}$; go to Step 4.2.

4.6 If α is a list whose first element is not the name of a function: let β be equal to the result of interpreting list α following Step 4.1 recursively.

Let *final-results-list* \leftarrow *final-results-list* \cup $\{\beta\}$.

Go to Step 4.2.

4.3 Problem Solving and Control

In solving a planning problem, PLANEX *selects* and *executes* various problem-solving operators. Controlling the execution of these operators is important because the effects of some operators usually affect the execution of other operators. This section describes in detail the structure and performance of the

system components responsible for *selecting* which feasible operators should be executed at each step in the planning process. The relationships among these components provide the system with the capability to solve problems in different domains. This versatility is illustrated in the next chapter.

During the development of PLANEX, several problem-solving mechanisms were explored. The first prototype system for excavation tasks placed all control decisions in the procedural codes of the operators. The system was inflexible in adapting its problem-solving behavior when obtaining the solution of a problem. In developing the initial architecture of CONSTRUCTION PLANEX, the need to separate domain operators from control operators was recognized. A message interface was proposed as a means to communicate with both types of operators. However, it was not clear what type of messages should be included in such an interface and how overall control would be accomplished. The first CONSTRUCTION PLANEX prototype included a set of well-defined domain operators for planning. In this system, control was provided through a menu-driven interface. The final PLANEX architecture incorporates both explicit control operators and menus.

4.3.1 Overview of Control Behavior

There are numerous alternatives for controlling the problem-solving operators. In systems that use a pure forward-chaining control strategy, problem-solving operators are executed whenever enough information to invoke them is available. Operator dependencies are implicitly expressed as a function of the availability of data in the context. Control is *opportunistic* and *myopic*. In contrast, *strategic* plan formulation systems such as NOAH (see p. 23) use declarative information about problem-solving actions to analyze operator interactions and generate plans. Although these systems have not been applied to the problem of controlling the execution of the problem-solving operators in an expert system, providing such a declarative representation of these operators seems promising.

The control mechanism of PLANEX combines opportunistic and strategic elements by using the *hybrid* model for process planning described in Section 3.3. In this model, the description of the problem-solving operators includes the input data they require (their *preconditions*) and the results they produce (their *effects*). Control alternates between the two phases shown in Figure 4-6:

- a *planning* phase in which operator execution is simulated on the basis of information about the preconditions and *predictable* effects of the operators; and
 - an *execution* phase in which operators are executed and their *unpredictable* effects are recorded.
-

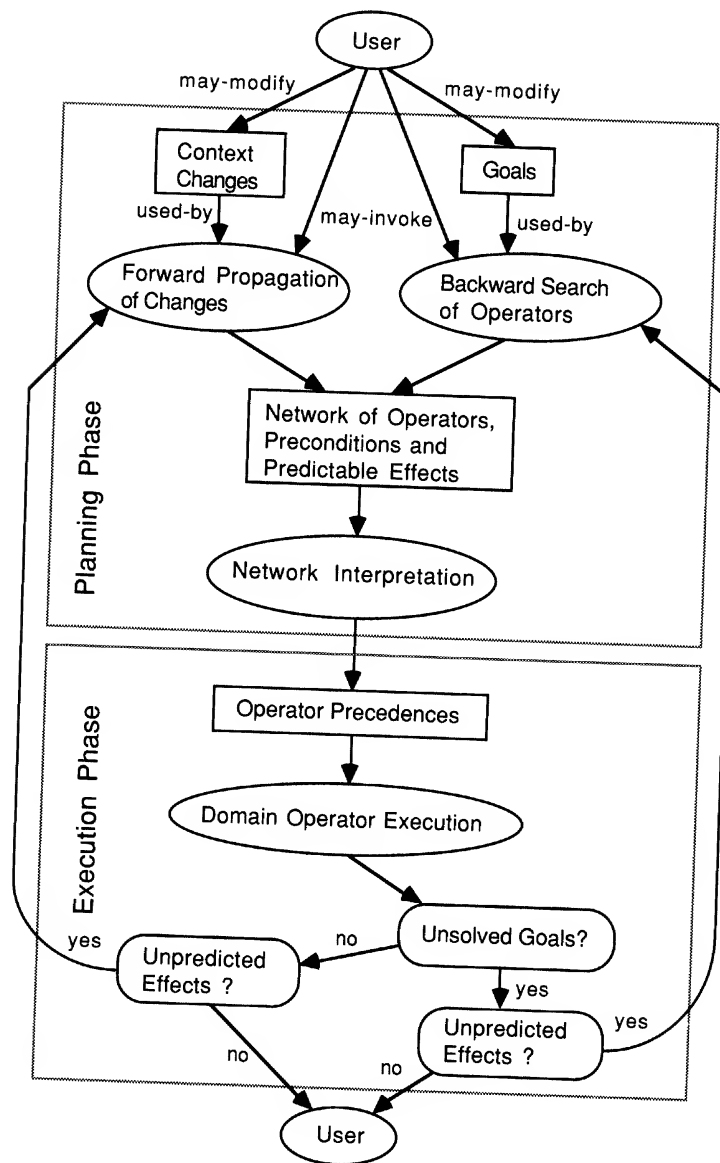


Figure 4-6. Control Behavior of PLANEX

The rationale for alternating between the two phases is based on the nature of the operators themselves. If the effects of the operators could be fully determined before they are executed, the planning phase need be performed only once. The system would know all of the operators that have to be executed and it could determine an order for their execution. In reality, some of the effects can only be determined *after* some operators are executed. For example, consider the effects of an operator responsible for allocating machines to activities. When the operator is applied to an activity, the name of the machine is stored in the *technology* slot of the activity. This effect is *predictable* because it can be described by identifying the locations in the context (i.e., which slot of which object) where the operator stores data. The operator also stores the name of the activity with the machine. However, the name of this machine (i.e., the target storage location for the result) is known only after the operator has been executed. This effect is *unpredictable* (i.e., the slot is known but the target object is not known until the operator has been executed) and is inserted in the agenda schema after the operator is executed to determine if it influences the execution of other operators. Thus, the controller must alternate between the two phases until the planning phase is executed with complete information so that no additional effects are produced.

The planning phase operates in two modes:

- *forward*, in which the system identifies those operators whose input data has changed and which have to be executed in order to maintain the consistency of the context; and
- *backward*, in which the system searches to find a series of partially ordered operators to achieve desired effects.

The need to simulate the execution of operators in the forward mode is illustrated in Figure 4-7. Suppose that the user introduces *change-1* in the context and this permits the application of operators *operator-1* and *operator-2*. In a purely opportunistic control strategy, either of these two operators may be executed first. Assume the scheduler chooses *operator-1*. This causes a new change, *change-2*, to be asserted in the context. If recent assertions are considered more important, subsequent operators activated by *change-2* would be executed next. At some later point, *operator-2* is executed and *change-3* is asserted. However, this change requires the execution of *operator-1* and its subsequent operators for a second time. In order to eliminate these cyclic operator invocations, operator execution is first simulated and precedences are identified. Then operators are executed in an order that avoids cyclic computations.

Backward search for operators is needed when the user requests information that is available only after executing more than one operator. PLANEX has to search for sequences of operators that provide this information. If several pieces

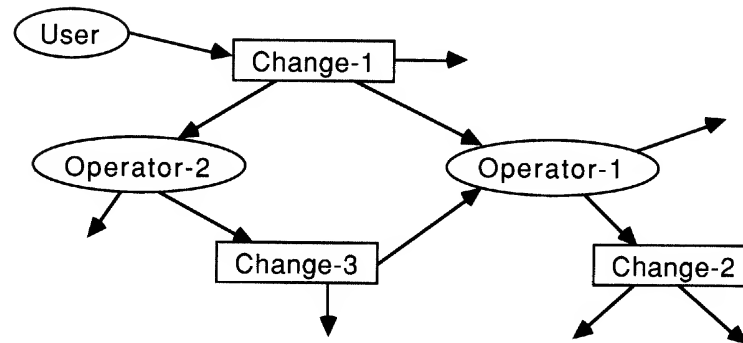


Figure 4-7. A Network of Operators and Changes

of information are requested, conflicts among different operator execution sequences must be considered. In the backward search for operator sequences, PLANEX uses a generalization of a network planning algorithm originally developed to solve problems in the blocks-world domain. As shown in the next chapter, this algorithm can solve problems with different types of goal interactions, such as the double-cross conflict problem described by Corkill [15].

4.3.2 Architectural Components Used for Control

The architecture of PLANEX incorporates four components used to implement the control behavior described above:

- *Domain Operator Schemas (DOSs)* that represent preconditions and effects of domain operators;
- An *Agenda* that stores: (1) pending (*operator object*) pairs similar to the *KSARs* in the *To-Do-Set* of OPM (see p. 32); (2) a list with precedences among domain operators; and (3) lists of context changes or goals to be achieved;
- *Control Operators* that: (1) determine the sequence in which domain operators should be executed; (2) modify the agenda; and (3) execute domain operators; and
- A *Menu-Driven Interface* that lets the user insert or delete context changes and goals, or modify the order in which operators are executed.

Figure 4-8 shows the relationships among the system components that control the execution of PLANEX. *Control* operators determine the sequence in which domain operators are executed by analyzing the declarative knowledge stored in each operator's corresponding DOS. Each DOS stores information about: (1) the location in the context which holds input data for a domain operator; (2) the name of the KS required by the operator; and (3) the location in

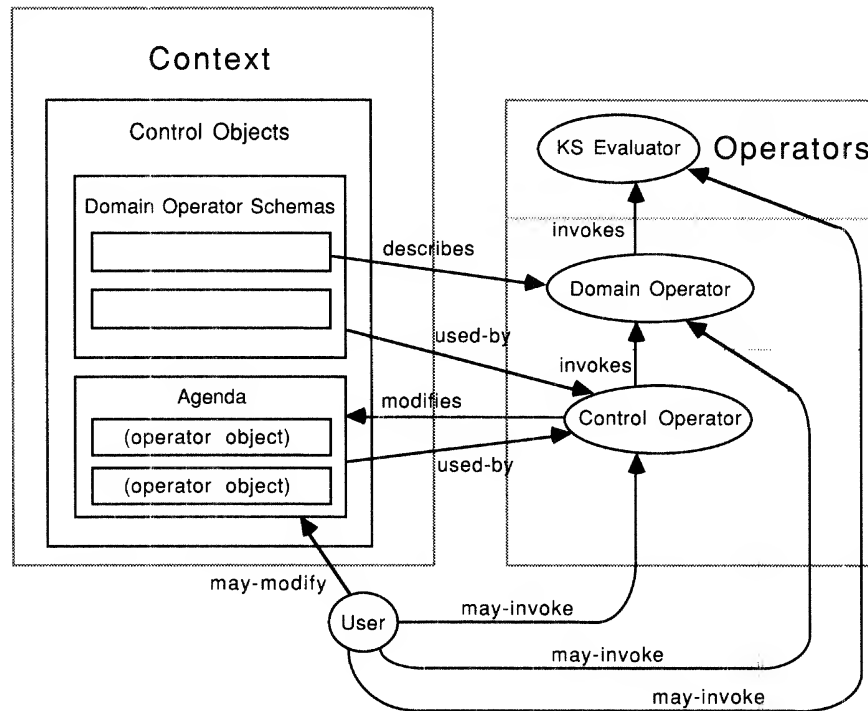


Figure 4-8. Relationships of Control Operators, Domain Operators and Control Objects

the context where the results computed by the operator will be stored. Control information stored in the agenda is modified and retrieved by control operators. Some control operators plan execution sequences of domain operators while others are responsible for executing the *(operator object)* pairs and recording unpredictable effects. Via the *menu-driven interface*, the user may also modify the information in the agenda (e.g., by inserting goals to be satisfied or changes to be propagated) or invoke the execution of specific domain or control operators.

4.3.3 Declarative Representation of Domain Operators

In PLANEX, an operator retrieves input data from *input objects* and stores results in the set of *output objects*. The attributes of the input and output objects which are related to the execution of the operator provide the basic information needed to control the execution of the problem-solving operators. This information is represented in the Domain Operator Schema (DOS) of the operator. Figure 4-9 shows the DOS of an operator that estimates the crew cost of construction activities. The slots of this schema are:

```

(defschema Get-Crew-Cost
  (is-a                operator)
  (domain-type         project-activity)
  (application-object   current-object)
  (input-objects        current-object <crew> <crew>
                        current-object current-object)
  (input-slots          technology normal-cost overtime-cost
                        normal-hours overtime-hours)
  (input-bindings       <crew> nil nil nil nil)
  (input-cond-types     filled filled filled filled filled)
  (output-objects       current-object current-object)
  (output-slots          tot-cost-crew $-crew/day)
  (output-bindings       nil nil)
  (output-predictable   yes yes)
  (output-effect-type   fill fill))

```

Figure 4-9. Schema Representation of a Domain Operator

- *is-a* identifies the DOS as an instance of an operator schema.
- *domain-type* indicates that the operator is applied to activity objects.
- *application-object* specifies the name of the object to which the operator is applied (e.g., a specific project activity).
- *input-objects* indicate the objects that are the sources of data for the operator. In this case, the *input-objects* are the *current-object* (i.e., the object to which the operator is applied), and the object associated with the binding variable *<crew>*.
- *input-slots* specify which slots of the corresponding *input-objects* contain data for the operator. For example, the first value indicates that the value of the *technology* slot of the *current-object* is used by the operator.
- *input-bindings* specify the bindings of the corresponding *input-objects*. In this example, the variable *<crew>* is bound to the value of the *technology* slot (i.e., the name of the technology used to perform the activity) of the *current-object*.
- *input-cond-types* specify if the input slot must be “filled” or “erased” before the operator is executed. Empty slots are considered erased.
- *output-objects* specify the objects that contain the results generated by the operator.
- *output-slots* specify the slots of the corresponding *output-objects* that store the results of the operator. In the example, the value of the *tot-cost-crew* and *\$-crew/day* slots of the project activity to which the operator is applied are modified by the operator.
- *output-bindings* specify the bindings of corresponding *output-objects*. No binding variables are used as output objects in this example.
- *output-predictable* specifies if an effect is predictable (the corresponding output object can be determined *before* the operator is executed). Allowable

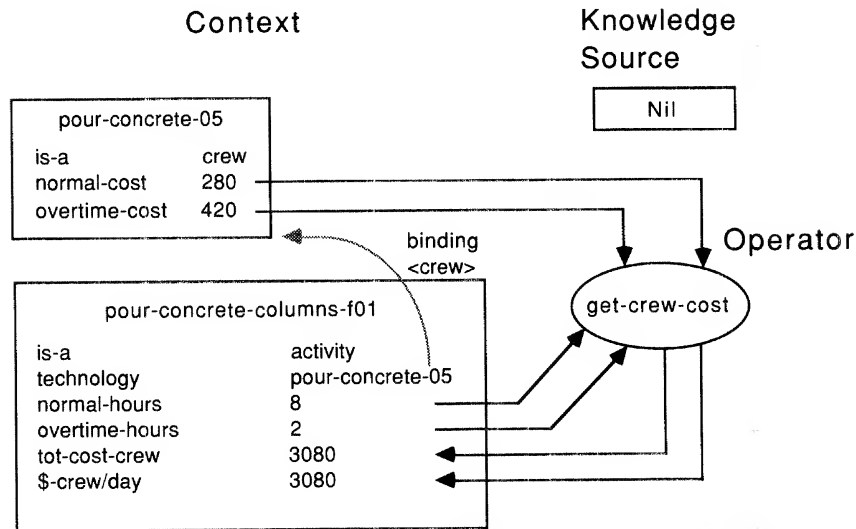


Figure 4-10. Application of a Domain Operator to a Context Object

values of this slot are "yes" or "no". In the example, all of the effects are predictable.

- *output-effect-type* indicates if the operator *fills* or *erases* the value of the *output-slots* in the corresponding *output-objects*. In the example, the schema specifies that when the operator is applied, it fills the *tot-cost-crew* and *\$-crew/day* slots of the *project-activity* object to which it is applied.

The information in the schema is an *abstract* description of the operator's input and output objects, and is interpreted by the system to determine specific information when the operator is applied. Consider the application of the operator *get-crew-cost* to the construction activity *pour-concrete-columns-f01*, as shown in Figure 4-10. The input objects specified in the schema are the *current-object* and the *<crew>*. These input objects correspond to the domain objects *pour-concrete-columns-f01* and *pour-concrete-05* (i.e., the value of the *technology* slot of the *current-object*). The operator uses the data in the *normal-hours* and *overtime-hours* slots of the activity object, and the *normal-cost* and *overtime-cost* slots of the crew object. Executing the operator modifies the values of the *tot-cost-crew* and *\$-crew/day* slots of the specific *project-activity* object.

4.3.4 Representation of Control Information

Control information is stored in the *agenda* schema. This information is generated (i.e., *posted*) and used by the control operators to determine the execution order of the domain operators. Figure 4-11 shows an example of control information stored in the *agenda* schema. The slots of the *agenda* are:

- *operator-queue* contains the operators to be executed. Each element has the form (*operator object*) indicating an *operator* which is to be applied to an *object*.
- *operator-precedences* are used to sort operators before they are executed. Each value is a pair ((*operator-1 object-1*) (*operator-2 object-2*)) that indicates that the first operator (e.g., *operator-1*) has to be executed before the second (e.g., *operator-2*).
- *goals* specify information to be computed. Goals are posted when the user requests object attribute values that have not yet been calculated. Each value has the form (*object slot type*) indicating that attribute *slot* of schema *object* is to be computed.
- *context-changes* describe the object and slot where a change is asserted and the type of change ("filled" or "erased"). Each value is a list (*object slot type*). Overwriting a slot value is considered a fill operation.
- *effect-operators* describe the results of an operator. Each value is a list ((*object slot type*) (*operator-1 object-1*) ... (*operator-n object-n*)) indicating that slot *slot* of object *object* may be "filled" or "erased" by applying one of the operators to its associated object (e.g., operator *operator-1* is applied to object *object-1*).
- *operator-preconditions* specify the preconditions of an operator. Each value is a list of the form ((*operator object*) ((*object-1 slot-1 type-1*) ... (*object-n slot-n type-n*))). This list defines the preconditions that must be satisfied before *operator* is applied to *object*. Preconditions are of the form (*object-i slot-i type-i*) indicating that *slot-i* of *object-i* be of *type-i*. Type is either "filled" or "empty".

The *effect-operators* and *operator-preconditions* slots of the *agenda* object are used both for forward propagation of changes and backward expansion of goals. In forward propagation, an *effect-operators* value indicates that executing the operator will assert a change in the context. In backward propagation, an *operator-preconditions* value is used to determine subgoals that must be satisfied.

```

(defschema agenda
  (operator-queue
    ((get-duration pour-concrete-columns-f01)
     (get-crew-cost pour-concrete-columns-f01)))
  (operator-precedences
    ((get-duration pour-concrete-columns-f01)
     (get-crew-cost pour-concrete-columns-f01)))
  (goals
    ((pour-concrete-columns-f01 tot-cost-crew filled)))
  (context-changes
    ((pour-concrete-05 normal-cost filled)
     (pour-concrete-05 overtime-cost filled)))
  (effect-operators
    ((pour-concrete-columns-f01 tot-cost-crew filled)
     (get-crew-cost pour-concrete-columns-f01)
     (pour-concrete-columns-f01 $-crew/day filled)
     (get-crew-cost pour-concrete-columns-f01)))
  (operator-preconditions
    ((get-crew-cost pour-concrete-columns-f01)
     (pour-concrete-columns-f01 normal-hours filled)
     (pour-concrete-columns-f01 overtime-hours filled)
     (pour-concrete-05 normal-cost filled)
     (pour-concrete-05 overtime-cost filled))))

```

Figure 4-11. Example Agenda Schema

4.3.5 Control Operators

PLANEX includes four control operators:

- *Forward Propagation Operator* (FPO) identifies which operators to execute due to changes introduced into the context;
- *Backward Search Operator* (BSO) finds sequences of domain operators that may achieve particular goals;
- *Network Interpretation Operator* (NIO) extracts operator precedences based on their preconditions and effects; and
- *Domain Operator Executor* (DOE) executes domain operators using information about operator precedences.

Control operators are algorithmic (i.e., they do not require a KS evaluation). The operator algorithms and the auxiliary procedures and data structures are discussed below.

4.3.5.1 Forward Propagation Operator Forward propagation of changes to context objects introduced by the user or domain operators is important in maintaining the consistency of process planning information. In PLANEX, context consistency is maintained by invoking the *Forward Propagation Operator*

(FPO). This control operator uses an algorithm based on the structure of the DOSs described above. The algorithm starts with a list of the changes to be propagated and creates a network of operators whose effects can be predetermined. Operator effects are used to simulate operator execution and to establish operator precedences. The network is expanded until all operators have been examined and no more predictable effects remain to be considered.

Before presenting the details of the algorithm for forward propagation, two auxiliary procedures are described:

- *Get Immediate Changes of an Operator.* This procedure is used to identify changes in context objects that are asserted when executing an operator. Changes are identified by examining the *output-objects* slot of the DOS associated with the operator. Only the predictable effects of the operator (those with *output-predictable* value "yes") are returned by this procedure. For example, the changes caused by applying the operator *get-crew-cost* of Figures 4-9 and 4-10 to the object *pour-concrete-columns-f01* would be:

```
( (pour-concrete-columns-f01 tot-cost-crew filled)
  (pour-concrete-columns-f01 $-crew/day filled) )
```

- *Get Immediate Operators of a Change.* This procedure is used to identify the operators that are activated (i.e., can now be executed) as a result of asserting a change in the context. Only those operators which contain the modified slot in the list of their *input-slots* and which have all of their preconditions filled with values may be activated by this assertion. For example, assume that the *normal-hours* slot of the activity object *pour-concrete-columns-f01* has been changed. Then the list of operators returned would include (assuming all the *input-slots* have values):

```
(get-crew-cost pour-concrete-columns-f01)
```

The procedure is quite complex when objects of different domain operators have slots of the same name. In this case, the information in the *input-type* slot is used to select only those objects belonging to the operator's domain. During execution, the procedure searches for binding values until a match with a specific object is found. For example, if the *overtime-cost* of a crew object *machine-1* is changed, the procedure searches for those activity objects with a value of "machine-1" in their *technology* slot.

The following data structures are used in the implementation of the FPO algorithm:

- *Queues of Operators.* Each member of the queue is composed of a pair of the form (*operator object*). There are two queues of operators:
 - *rem-queue-op* is a list of the names of those operators that are ready to be executed as a result of changes introduced in the context.
 - *phase-queue-op* contains the list of all the operators identified while executing the planning phase of the FPO.

- *Queues of Changes.* Each element in the queue has the form (*object slot type*), describing the *object* and *slot* where a change is asserted and the *type* of change ("filled" or "erased"), similar to the elements in the *context-changes* slot of the *agenda*. There are three queues of changes:
 - *rem-queue-chgs* contains changes that have not yet been considered in identifying the operators to be executed as a result of assertions in the context.
 - *phase-queue-chgs* contains the list of all changes that have been analyzed during the execution of the FPO.
 - *final-queue-chgs* contains the list of changes that do not cause the execution of any domain operator.
- *Network of Operators and Changes.* During the execution of the FPO, a network of operators and changes is created. Two lists are used to represent the network:
 - *op-prec-list* contains information about the immediate operators activated for particular context changes. The elements of this list have similar syntax to those in the *operator-preconditions* slot of the *agenda* schema. Each value is a list of the form ((*operator object*) ((*object-1 slot-1 type-1*) ... (*object-n slot-n type-n*))). The list defines the preconditions that must be true before *operator* is applied to *object*. Preconditions are of the form (*object-i slot-i type-i*) indicating that *slot-i* of *object-i* be of the designated type ("filled" or "empty").
 - *eff-op-list* contains information about predictable effects for particular operators. The elements of this list have similar syntax to the elements in the *effect-operators* slot of the *agenda* schema. Each value is a list ((*object slot type*) (*operator-1 object-1*) ... (*operator-n object-n*)). The list indicates that the slot *slot* of the *object* is "filled" or "erased" by applying one of the operators to its associated object (e.g., *operator-i* applied to *object-i*).

The algorithm for the FPO is detailed below. It has three steps: (1) initialization of information from the *agenda*; (2) recursive expansion of the network of operators and changes; and (3) storage of result information in the *agenda*.

Algorithm for the Forward Propagation Operator

Step 1. Initialize

- 1.1 Create empty lists for *rem-queue-op*, *phase-queue-op*, *phase-queue-chgs*, and *final-queue-chgs*.
Retrieve values from the *agenda* schema: Assign:
 (1) *op-prec-list* the value of the *operator-preconditions* slot;
 (2) *eff-op-list* the value of the *effect-operators* slot; and
 (3) *rem-queue-chgs* be the value of the *context-changes* slot.

Step 2. Expand Network of Operators and Changes

- 2.1 If *rem-queue-chgs* is empty: go to Step 3.
Let λ be the first change in *rem-queue-chgs*.
Let *rem-queue-chgs* \leftarrow *rem-queue-chgs* $\setminus \{\lambda\}$.
- 2.2 If $\lambda \in$ *phase-queue-chgs*: go to Step 2.1.
Use procedure *Get Immediate Operators of a Change* to find the list of immediate operators *O* caused by change λ .
Let *phase-queue-chgs* \leftarrow *phase-queue-chgs* $\cup \{\lambda\}$.
- 2.3 If *O* is empty: let *final-queue-chgs* \leftarrow *final-queue-chgs* $\cup \{\lambda\}$; go to Step 2.1.
Let *rem-queue-op* \leftarrow *O*.
- 2.4 Create links from change λ to each operator in *O* and add them to the corresponding elements of *op-prec-list*.
- 2.5 If *rem-queue-op* is empty: go to Step 2.1.
Let *o* be the first operator in *rem-queue-op*.
Let *rem-queue-op* \leftarrow *rem-queue-op* $\setminus \{o\}$.
- 2.6 If *o* \in *phase-queue-op*: go to Step 2.4.
Use procedure *Get Immediate Changes of an Operator* to find the list of immediate changes Λ caused by operator *o*.
- 2.7 Create links from operator *o* to changes in Λ and add them to the *eff-op-list*.
Let *phase-queue-op* \leftarrow *phase-queue-op* $\cup \{o\}$.
- 2.8 Let *rem-queue-chgs* \leftarrow *rem-queue-chgs* $\cup \Lambda$.
Go to Step 2.5.

Step 3. Modify the Agenda

- 3.1 Modify the value of the following slots of the *agenda*: Store:
 (1) *op-prec-list* in the *operator-preconditions* slot;
 (2) *eff-op-list* in the *effect-operators* slot;
 (3) *final-queue-chgs* in the *context-changes* slot; and
 (4) *phase-queue-op* in the *operator-queue* slot.

4.3.5.2 Backward Search Operator In solving a problem, PLANEX not only propagates the effects of planning decisions, but also finds sequences of operators to achieve the desired effects. These effects (goals) are expressed in terms of object attributes that the system is to compute (e.g., find the *duration* and *cost* of an activity). The system derives sequences of problem-solving operators that transform the initial state (e.g., the *duration* and *cost* of the activity are unknown) into the desired state. PLANEX determines these sequences by invoking a *Backward Search Operator* (BSO) whose action is similar to some of the plan formulation systems described in the Chapter 2. This version has been modified to produce plans that have a minimum number of actions.

The algorithm of the BSO is a specialization of the AO* search procedure [80, p. 63]. The algorithm expands each final goal independently. Goal expansion yields an associated set of *potential* operators whose execution may require certain conditions be satisfied. These operators will be labeled *infeasible* if they are found to interfere with other operators in the same sequence, or *executable* if all their preconditions are satisfied. When an operator is labeled executable, all its predictable assertions are considered *solved* and the algorithm looks for other operators that become executable because of these changes. The expansion process continues until all goals have been expanded into a network of executable and infeasible operators.

Before presenting the details of the algorithm, three auxiliary procedures are described:

- *Get Possible Operators to Achieve a Goal.* This procedure searches the *output-effects* slot of the DOSs and returns a list of the operators that may be executed to fill or erase the value of a specific slot in a frame. For example, if the goal is to fill the slot *tot-cost-crew* of the *pour-concrete-columns-f01* object, this procedure would return:
 ((get-crew-cost pour-concrete-columns-f01))
- *Get Goals Required to Execute an Operator.* This procedure examines the DOS associated with the operator and returns a list of the operator's preconditions that are not satisfied in the current state. In this process, binding variables are evaluated sequentially. For example, suppose that this procedure is employed to find the unsatisfied preconditions of the operator (*get-crew-cost pour-concrete-columns-f01*). If the *normal-hours* and *overtime-hours* slots of the *pour-concrete-columns-f01* frame are empty, but the *normal-cost* and *overtime-cost* slots of the *pour-concrete-05* frame are filled, this procedure would return:
 ((pour-concrete-columns-f01 normal-hours filled)
 (pour-concrete-columns-f01 overtime-hours filled))
- *Check the Feasibility of an Operator.* This procedure determines if an operator is infeasible or executable. An operator is infeasible when it negates a necessary precondition of a successor operator in the network. The

feasibility criterion is similar to the *necessary truth* criterion of TWEAK [12]. Figure 4-12 shows the application of this criterion. In the figure, arrows indicate the following relationships:

- a *precondition* is indicated with an unlabeled arrow directed from a condition to an operator;
- an *assertion* is indicated with an arrow labeled with a plus sign (\oplus) directed from an operator to a condition; and
- a *negation* is indicated with an arrow labeled with a minus sign (\ominus) directed from an operator to a condition.

A negation occurs when an operator yields an effect on a slot of an object that contradicts the effect of another operator on the same object. The example network indicates that *operator-1* and *operator-2* are possible operators to achieve *goal-1*. In order to execute *operator-1*, subgoals *goal-2*, *goal-3*, *goal-5* and *goal-6* have to be asserted. The procedure labels *operator-5* as “infeasible” because it negates the precondition *goal-3* of *operator-1* when trying to assert a subgoal of the same operator (*goal-6* needed by *operator-3* to assert *goal-2* of *operator-1*). It does not label *operator-4* as “infeasible” because it does not negate any other precondition of *operator-1*.

The following data structures are used in the search procedure of the BSO:

- *Queues of Operators*. Each member of the queue is composed of a pair of the form (*operator object*). The four operator queues are:
 - *rem-queue-op* contains the names of those operators whose immediate subgoals have not yet been determined.
 - *phase-queue-op* contains the list of all operators identified by the BSO.
 - *inf-op-list* contains the names of the operators that have been labeled “infeasible”.
 - *exec-op-list* contains the names of the operators that have been labeled “executable”.
- *Queues of Goals*. Each element in the queue has the form (*object slot type*) indicating that the *slot* of *object* is to be “filled” or “erased” by the system. There are four queues of goals:
 - *rem-queue-goals* contains goals that have not yet been analyzed to identify which operators may be used to achieve them.
 - *phase-queue-goals* contains the list of all goals that have been analyzed.
 - *phase-solved-goals* contains goals that have been successfully expanded.
 - *phase-unsolved-goals* contains goals that cannot be satisfied in the current planning phase.

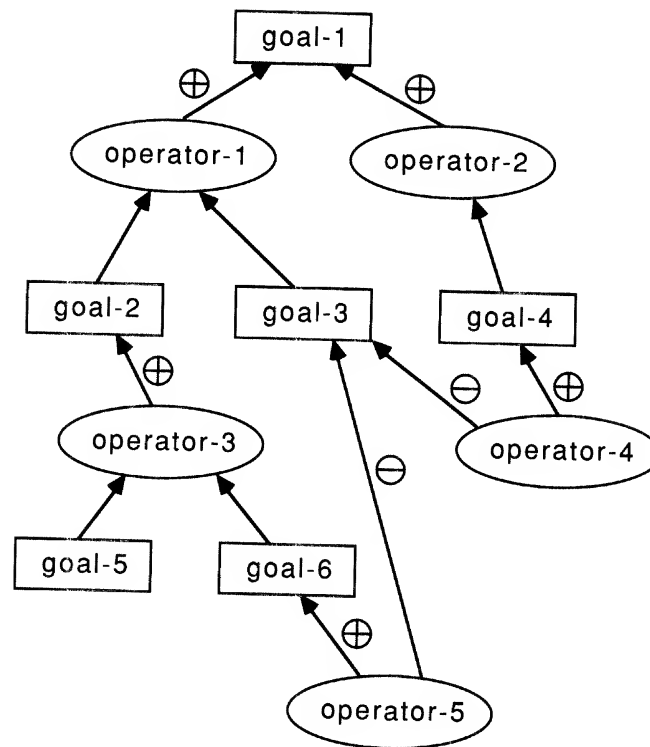


Figure 4-12. Operator Feasibility in Backward Search

- *Network of Operators and Goals.* Two lists are used to represent the links between goals and operators:
 - *op-prec-list* stores links between an operator and the goals associated with its preconditions. Each value is a list of the form *(operator object)* *((object-1 slot-1 type-1) ... (object-n slot-n type-n))*. The list defines the preconditions that must be true before *operator* is applied to *object*. Preconditions are of the form *(object-i slot-i type-i)* indicating that *slot-i* of *object-i* be of the designated type ("filled" or "empty").
 - *eff-op-list* stores links between an operator and the goals that the operator asserts. Each value is a list of the form *(operator slot type)* *((operator-1 object-1) ... (operator-n object-n))*. The list indicates the slot *slot* of frame *object* is "filled" or "erased" by applying one of the operators to its associated object (e.g., *operator-i* is applied to *object-i*).

The algorithm for the BSO is detailed below. It has five steps: (1) initialization and *agenda* information retrieval; (2) goal expansion by iden-

tifying possible operators to achieve the goals; (3) backtracking to delete infeasible operators and mark goals unachievable; (4) attainment of goals; and (5) storage of remaining goals in the *agenda* schema.

Algorithm for the Backward Search Operator

Step 1. Initialize

- 1.1 Create empty lists for *op-prec-list*, *eff-op-list*, *phase-queue-goals*, *phase-solved-goals*, *phase-unsolved-goals*, *phase-queue-op*, *rem-queue-op*, *exec-op-list* and *inf-op-list*.
Let *rem-queue-goals* and *phase-queue-goals* be the initial list of goals to be achieved, taken from the *agenda*.

Step 2. Expand a Goal

- 2.1 If *rem-queue-goals* is empty: go to Step 5.
Let λ be the first goal in *rem-queue-goals*.
Let $\text{rem-queue-goals} \leftarrow \text{rem-queue-goals} \setminus \{\lambda\}$.
Let $\text{phase-queue-goals} \leftarrow \text{phase-queue-goals} \cup \{\lambda\}$.
- 2.2 If $\lambda \in \text{phase-solved-goals}$: go to Step 2.1.
Use procedure *Get Possible Operators to Achieve a Goal* to find the list of possible operators *O* to achieve goal λ .
- 2.3 Use procedure *Check the Feasibility of an Operator* to eliminate from *O* those operators that are infeasible.
If *O* is not empty: go to Step 2.4.
Let $\text{phase-unsolved-goals} \leftarrow \text{phase-unsolved-goals} \cup \{\lambda\}$.
Go to Step 3.
- 2.4 Create links from each operator in *O* to goal λ and add them to *eff-op-list*.
Let $\text{rem-queue-op} \leftarrow O$.
Let $\text{phase-queue-op} \leftarrow \text{phase-queue-op} \cup O$.
- 2.5 If *rem-queue-op* is empty: go to Step 2.1.
Let *o* be the first operator in *rem-queue-op*.
Let $\text{rem-queue-op} \leftarrow \text{rem-queue-op} \setminus \{o\}$.
- 2.6 If $o \in \text{exec-op-list}$: let $\text{phase-solved-goals} \leftarrow \text{phase-solved-goals} \cup \{\lambda\}$; go to Step 4.
- 2.7 Use procedure *Get Goals Required to Execute an Operator* to find the list of immediate preconditions Λ for operator *o*.
Create links from each condition in Λ to operator *o* and add them to *op-prec-list*.
- 2.8 Let $\text{rem-queue-goals} \leftarrow \text{rem-queue-goals} \cup \Lambda$.
Go to Step 2.1.

Step 3. Backtrack from a Goal

- 3.1 Let $\Lambda_1 \leftarrow \{\lambda\}$, where λ is the unsolvable goal.
- 3.2 If Λ_1 is empty: go to Step 2.1.
Let λ_1 be the first goal in Λ_1 .
Let $\Lambda_1 \leftarrow \Lambda_1 \setminus \{\lambda_1\}$.
- 3.3 Let O be the operators in *op-prec-list* requiring precondition λ_1 .
- 3.4 If O is empty: go to Step 3.2.
Let o be the first operator in O .
Let $O \leftarrow O \setminus \{o\}$.
- 3.5 Let Λ_2 be the goals of *eff-op-list* achievable by applying operator o .
Let *phase-queue-op* \leftarrow *phase-queue-op* $\setminus \{o\}$.
- 3.6 Remove from *eff-op-list* those elements associated with operator o .
- 3.7 If Λ_2 is empty: go to Step 3.2.
Let λ_2 be the first goal in Λ_2 .
Let $\Lambda_2 \leftarrow \Lambda_2 \setminus \{\lambda_2\}$.
- 3.8 Let X be the set of operators that achieve λ_2 that remain in *eff-op-list*.
- 3.9 If X is non-empty: go to Step 3.7.
Let *phase-unsolved-goals* \leftarrow *phase-unsolved-goals* $\cup \{\lambda_2\}$.
Let $\Lambda_1 \leftarrow \Lambda_1 \cup \{\lambda_2\}$.
Go to Step 3.7.

Step 4. Achieve a Goal

- 4.1 Let $\Lambda_1 \leftarrow \{\lambda\}$, where λ is the solved goal.
- 4.2 If Λ_1 is empty: go to Step 2.1.
Let λ_1 be the first goal in Λ_1 .
Let $\Lambda_1 \leftarrow \Lambda_1 \setminus \{\lambda_1\}$.
- 4.3 Let O be the operators in *op-prec-list* requiring precondition λ_1 that do not belong to *exec-op-list*.
- 4.4 If O is empty: go to Step 4.2.
Let o be the first operator in O .
Let $O \leftarrow O \setminus \{o\}$.
- 4.5 Let Λ_3 be the preconditions of operator o .
If all elements of $\Lambda_3 \in$ *phase-solved-goals* or are true in the context: go to Step 4.6.
Go to Step 4.2.

- 4.6 Let Λ_2 be the goals in *eff-op-list* achievable by applying operator o .
 Let $\text{phase-solved-goals} \leftarrow \text{phase-solved-goals} \cup \Lambda_2$.
 Let $\Lambda_1 \leftarrow \Lambda_1 \cup \Lambda_2$.
 Go to Step 4.2.

Step 5. Modify the Agenda

- 5.1 Modify the value of the following slots of the *agenda*: Store:
 - (1) *op-prec-list* in the *operator-preconditions* slot;
 - (2) *eff-op-list* in the *effect-operators* slot;
 - (3) *phase-unsolved-goals* in the *goals* slot; and
 - (4) *phase-queue-op* in the *operator-queue* slot.

4.3.5.3 Network Interpretation Operator After the planning phase is performed by the FPO or the BSO, the *agenda* contains a network of operators, preconditions and predictable effects in the *operator-preconditions* and *effect-operators* slots. The operators in the *operator-queue* represent all those tasks to be performed in order to maintain the consistency of the context or to achieve the desired goals. These operators are not independent because the effects of some operators are the preconditions of others.

The need to interpret the *agenda* information before executing these operators is illustrated in the network of Figure 4-7. Assume values for all objects are available (including *change-3*) and *operator-1* is executed before *operator-2*. After execution of both operators, the context would be inconsistent because *operator-2* asserts a value for *change-3* that was not used in the execution of *operator-1*. If *change-3* initially was not available in the context, *operator-2* would have to be executed before *operator-1* in order to provide *operator-1* with its required preconditions.

In PLANEX, the network of operators and conditions is interpreted by applying the *Network Interpretation Operator* (NIO). The NIO analyzes the relationships between each operator of the network and its neighbors. Two operators are considered neighbors when their effects or preconditions have at least one element in common. The criteria used by the algorithm of the NIO to identify operator precedences is shown in Figure 4-13. The following relationships are possible between two neighboring operators, *operator-1* and *operator-2*:

1. A precondition of *operator-2* is asserted by *operator-1*. *Operator-2* should be executed after *operator-1* asserts the condition.
2. A condition is a precondition of both operators. No operator precedence is required.
3. A condition is asserted by both operators. No operator precedence is required.
4. A precondition of *operator-2* is negated by *operator-1*. *Operator-2* should be executed before *operator-1* negates the condition.

Case	Neighbor Operators	Operator Precedence
1		
2		None
3		None
4		
5		None
6		None

Figure 4-13. Identification of Operator Precedences

5. A condition is negated by one operator and asserted by the other. No operator precedence is required because the condition does not affect the execution of either of the two operators.
6. A condition is negated by both operators. No operator precedence is required.

The algorithm of the NIO uses the same data structures as the FPO and BSO.

- *Queues of Goals.* Each element in the queue has the form (*object slot type*) indicating that the *slot* of *object* is to be “filled” or “erased” by the system. There is one goal queue:
 - *rem-queue-goals* contains goals that have not yet been analyzed to identify which operators may be used to achieve them;
- *Network of Operators and Goals.* Three lists are used to represent the links between goals and operators:
 - *op-prec-list* stores links between an operator and the goals associated with its preconditions. Each value is a list of the form (*operator object*) ((*object-1 slot-1 type-1*) ... (*object-n slot-n type-n*)). The list defines the preconditions that must be true before *operator* is applied to *object*. Preconditions are of the form (*object-i slot-i type-i*) indicating that *slot-i* of *object-i* be of the designated type (“filled” or “empty”).
 - *eff-op-list* stores links between an operator and the goals that are asserted by the operator. Each value is a list (*operator slot type*) ((*operator-1 object-1*) ... (*operator-n object-n*)). The list indicates the slot *slot* of frame *object* is “filled” or “erased” by applying one of the operators to its associated object (e.g., *operator-i* is applied to *object-i*).
 - *link-op-list* stores information about precedences between pairs of operators. The list is composed of a pair of elements of the form ((*operator-1 object-1*) (*operator-2 object-2*)), indicating that *operator-1* should be applied to schema *object-1* before *operator-2* is applied to *object-2*.

The algorithm is detailed below. It has three steps: (1) *agenda* information retrieval; (2) determination of operator precedences; and (3) storage of results in the *agenda*.

Algorithm for the Network Interpretation Operator

Step 1. Initialize

- 1.1 Create an empty list for *link-op-list*.
 Retrieve values from the *agenda*: Assign:
 - (1) *op-prec-list* the value of the *operator-preconditions* slot;
 - (2) *eff-op-list* the value of the *effect-operators* slot; and
 - (3) *rem-queue-op* the value of the *operator-queue* slot.

Step 2. Find Operator Precedences

- 2.1 If *rem-queue-op* is empty: go to Step 3.
 Let o be the first operator in *rem-queue-op*.
 Let $\text{rem-queue-op} \leftarrow \text{rem-queue-op} \setminus \{o\}$.
- 2.2 Let Λ be the list of preconditions of operator o in *op-prec-list*.
- 2.3 If Λ is empty: go to Step 2.1.
 Let λ be the first element of Λ .
 Let $\Lambda \leftarrow \Lambda \setminus \{\lambda\}$.
- 2.4 Find the set of operators O^+ in *eff-op-list* that assert condition λ .
- 2.5 If O^+ is empty: go to Step 2.7.
 Let o^+ be the first element of O^+ .
 Let $O^+ \leftarrow O^+ \setminus \{o^+\}$.
- 2.6 Let $\text{link-op-list} \leftarrow \text{link-op-list} \cup \{(o^+ o)\}$.
 Go to Step 2.5.
- 2.7 Find the set of operators O^- in *eff-op-list* that negate condition λ .
- 2.8 If O^- is empty: go to Step 2.3.
 Let o^- be the first element of O^- .
 Let $O^- \leftarrow O^- \setminus \{o^-\}$.
- 2.9 Let $\text{link-op-list} \leftarrow \text{link-op-list} \cup \{(o o^-)\}$.
 Go to Step 2.8.

Step 3. Modify the Agenda

- 3.1 Store the value of *link-op-list* in the *operator-precedences* slot of the *agenda*.

4.3.5.4 Domain Operator Executor Operator precedence information is used by the *Domain Operator Executor* (DOE) when invoking domain operators. The DOE executes operators in *topological order*⁷ and stores unpredictable effects in the *context-changes* slot of the *agenda*. The algorithm uses one auxiliary procedure:

- *Get Unpredictable Effects of an Operator*. This procedure is used to identify the set of unpredictable effects of an operator, similar to the procedure used to get the immediate changes of an operator (see p. 110). The procedure returns those effects that have a *output-predictable* value equal to “no” in the DOS associated with the operator.

The DOE algorithm uses some of the data structures described previously.

⁷ An operator is applied only after all its preceding operators have been executed.

- *Queues of Changes.* Each element in the queue has the form (*object slot type*), describing the *object* and *slot* where a change is asserted and the *type* of change (“filled” or “erased”), similar to the elements in the *context-changes* slot of the *agenda*. There is one change queue:
 - *phase-queue-chgs* contains the list of all changes that have been analyzed during the execution of the FPO.
- *Queues of Operators.* Each member of the queue is composed of a pair of the form (*operator object*). There is one operator queue:
 - *rem-queue-op* is a list of the names of those operators that are ready for execution as a result of changes introduced in the context.
- *Network of Operators and Goals.* One list is used to represent the links between goals and operator.
 - *link-op-list* stores information about precedences between pairs of operators. The list is composed of a pair of elements of the form ((*operator-1 object-1*) (*operator-2 object-2*)), indicating that *operator-1* should be applied to schema *object-1* before *operator-2* is applied to *object-2*.

The algorithm follows. It has three steps: (1) initialization; (2) sorting and execution of operators; and (3) storage of results in the *agenda*.

Algorithm for the Domain Operator Executor

Step 1. Initialize

- 1.1 Retrieve values from the *agenda*: Assign:
 - (1) *link-op-list* the value of the *operator-precedences* slot;
 - (2) *phase-queue-chgs* the value of the *context-changes* slot; and
 - (3) *rem-queue-op* the value of the *operator-queue* slot.

Step 2. Sort and Execute Operators

- 2.1 Topologically sort the *rem-queue-op* using the precedence information stored in *link-op-list*.
- 2.2 If *rem-queue-op* is empty: go to Step 3.
 Let *o* be the first element of *rem-queue-op*.
 Let $\text{rem-queue-op} \leftarrow \text{rem-queue-op} \setminus \{o\}$.
- 2.3 Execute domain operator *o*.
- 2.4 Use procedure *Get Unpredictable Effects of an Operator* to find the unpredictable changes Λ of operator *o*.
 Let $\text{phase-queue-chgs} \leftarrow \text{phase-queue-chgs} \cup \Lambda$.
 Go to Step 2.2.

Step 3. Modify the Agenda

3.1 Store the value of *phase-queue-chgs* in the *context-changes* slot of the *agenda*.

4.4 User Interaction

PLANEX is intended to provide a framework for developing *user assistants*. This requires the user be permitted to modify system components, invoke system operations, override planning decisions and request information about planning results. A user assistant is desirable because:

- there are process planning problems whose solution requires the direct participation of the user;
- creative planning involves common-sense knowledge that is difficult to incorporate into an automated system; and
- the user is responsible for adapting the system to different application domains.

Other reasons for not entirely eliminating human interaction and control in expert system applications are discussed by Waterman [107, p. 13].

In PLANEX, interaction is provided through several mechanisms which are deemed fundamental for achieving the desired generality, flexibility and transparency of the architecture. Examples of such interaction mechanisms are *menus* for invoking task-specific operators and interactive graphical displays for modifying the information stored in context objects.

During the development of PLANEX, support for user interaction has been repeatedly improved. In the first prototype for excavation tasks, user interaction was very limited. The only way to modify context objects was to use a set of primitive manipulation functions provided by the frame implementation language, such as the function to create a slot in a frame. Initially, no mechanisms to alter control decisions or knowledge were available. When this prototype evolved into CONSTRUCTION PLANEX, effective mechanisms for acquiring and updating the required domain-specific knowledge were incorporated into the system architecture. In addition, functions for producing reports and displays of context information were developed. Some of these mechanisms were generalized and restructured and now exist as components of PLANEX.

Figure 4-14 shows the different types of user interaction mechanisms available in PLANEX. There are mechanisms to:

- *modify* the information stored in context objects, the knowledge used by operators, and the manner in which operators use this knowledge. The user may modify any of the following system components:
 - *knowledge sources*, by using the KNOWLEDGE SOURCE ACQUISITION MODULE (KSAM) described in Section 4.4.1;

MODIFY				
<i>Mechanism</i>	<i>Knowledge Source</i>	<i>Domain Object</i>	<i>Control Object</i>	<i>Domain Operator</i>
KSAM	•			
Interactive Graphics		•		
Questions to User		•		
Control Panel			•	
Editor and Language	•	•	•	•

INVOKE			
<i>Mechanism</i>	<i>KSE</i>	<i>Domain Operator</i>	<i>Control Operator</i>
Control Panel		•	•
Menu	•	•	
Editor and Language	•	•	•

REQUEST			
<i>Mechanism</i>	<i>Knowledge Source</i>	<i>Domain Object</i>	<i>Control Object</i>
KSAM	•		
Report Generator		•	
Output Graphics		•	
Explanation		•	
Control Panel			•
Editor and Language	•	•	•

Figure 4-14. Mechanisms for User Interaction

- *domain objects*, by using an *interactive graphical* environment such as the *GANTT Interactive Scheduler* of CONSTRUCTION PLANEX;
 - *control objects*, by using the CONTROL PANEL (CP) described in Section 4.4.3 or primitives of the frame implementation language; and
 - *domain operators*, by editing the files which contain the procedural codes or *Domain Operator Schemas* (DOSs) of the operators.
- *invoke* the execution of specific domain or control operators and control the order in which these operators are applied. The user may invoke the KNOWLEDGE SOURCE EVALUATOR (KSE), *domain operators* and *control operators* by using the CONTROL PANEL described in Section 4.4.3, by means of a *menu-driven interface*, or by invoking the operator directly using the implementation language.
 - *request* information about data produced by the system and the manner in which this data was computed. The user may request information about any of the following system components:
 - *knowledge sources*, via the KNOWLEDGE SOURCE ACQUISITION MODULE;
 - *domain objects*, via: (1) the REPORT GENERATOR that lets the user design output reports that summarize context information; (2) output graphics such as the project cost curves of CONSTRUCTION PLANEX; or (3) the explanation facilities provided by the system; and
 - *control objects*, via the CONTROL PANEL or primitives provided by the implementation language.

In the following sections, several of the user interaction mechanisms of PLANEX are described. Some of these mechanisms are generic (e.g., the mechanism for updating domain knowledge and the CONTROL PANEL) while others are dependent upon particular application domains (e.g., the *GANTT Interactive Scheduler* of CONSTRUCTION PLANEX). The following discussion explains how these mechanisms can be utilized in knowledge-based process planning systems developed with PLANEX. Since these are only the mechanisms from which a complete user interface is built, the following discussion does not illustrate the use of PLANEX. Overall control is provided through a set of menus, the CONTROL PANEL and predefined sequences of problem-solving operators. Chapter 6 details of an actual user interface (Section 6.2.4) and the use and operation of a PLANEX application (Section 6.3).

4.4.1 Knowledge Source Acquisition Module

In PLANEX, knowledge is represented in knowledge sources (KSs) (see Section 4.2). As described, each KS may be considered a generalized *decision table* that groups rules sharing conditions and actions. This external view of a KS as a *tabular* representation is easier to understand than its associated *schema*

implementation. The KNOWLEDGE SOURCE ACQUISITION MODULE (KSAM) [116] is an interactive editor which uses the tabular representation of KSs. The KSAM lets the user create, modify or delete KSs, and after changes are made, it translates the tabular representation into the appropriate schema representation used by the KNOWLEDGE SOURCE EVALUATOR (KSE). Figure 4-15 illustrates this process. The KSs created or updated with the KSAM can be saved in files for future use. Also, previously saved KSs are updated by loading them from files and translating them into their tabular form.

The display of the KSAM contains the six windows shown in Figure 4-16. Four main windows correspond to the four regions of a KS: conditions, actions, left-hand-sides of rules and right-hand-sides of rules. Lines in these windows delimit *cells*. The user may enter or modify cell values, scroll the windows, select menu options or type values. The two small windows at the bottom of the screen are the *Current KS* window, which displays the name of the KS being modified and the firing mechanism used by the KSE, and the *Input/Output* window that is used to edit cell values using a line editor. The KSAM provides many desirable capabilities in an interactive editing environment. For example, when editing a row in the *Conditions* window, the KSAM looks for the schema with the name indicated in the *Object* column and displays the names of its slots. This provides additional help to the user in defining domain knowledge.

4.4.2 Interactive Graphical Displays

Interactive graphical displays provide the user with a convenient mechanism for modifying the information stored in domain objects. An example of this type of user interaction mechanism is the GANTT *Interactive Scheduler* [116]. GANTT lets the user calculate, display and modify scheduling information for a particular set of activities through an interactive Gantt-chart. The design of GANTT is based on the work of Garman [37] on applications of computer graphics in solving scheduling and resource allocation problems. Garman showed that novices using an interactive scheduling program performed better than the most efficient scheduling algorithms available. GANTT incorporates an interactive Gantt-chart plus displays of resource profiles that are automatically updated when activities are manually rescheduled.

The operation of GANTT is shown in Figure 4-17. The program lets the user interactively modify a schedule via a Gantt-chart display, propagate these changes to all of the activities of the project and store project schedules in context schemas. Project activity networks are represented using the unified activity network model described in Section 3.3. With this representation, schedule changes are propagated using common shortest-path algorithms. After scheduling, an updated project chart is displayed. Additional schedule changes may be introduced. When the user is satisfied, this project schedule may be

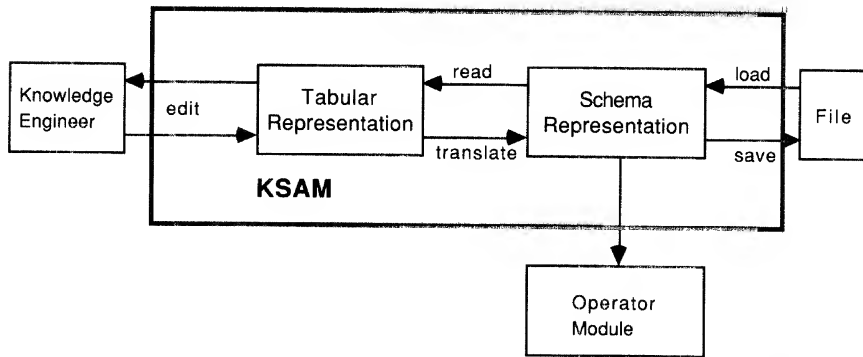


Figure 4-15. Illustration of the Knowledge Source Acquisition Process

Object	Slot	Operator	Value	KNOWLEDGE SOURCE ACQUISITION MODULE									
Conditions				Left Hand Side Rules									
Actions				Right Hand Side Rules									
INPUT/OUTPUT				Current KS									
Prompt?				KS-BUFFER					FIRST				
				Messages are posted here ...									

Figure 4-16. Screen Display of the KSAM

stored in the context schemas associated with the project activities. Alternative project schedules may be saved in files for later use.

A screen display of GANTT is shown in Figure 4-18. Schedule changes are made by pointing at one of the bars representing the activities and clicking a mouse button. Depending on the type of click used, an activity duration may be shortened or lengthened or the activity may be shifted in time. Via menu options, the user may specify milestones or modify activity precedences, time scales and resource trigger levels.

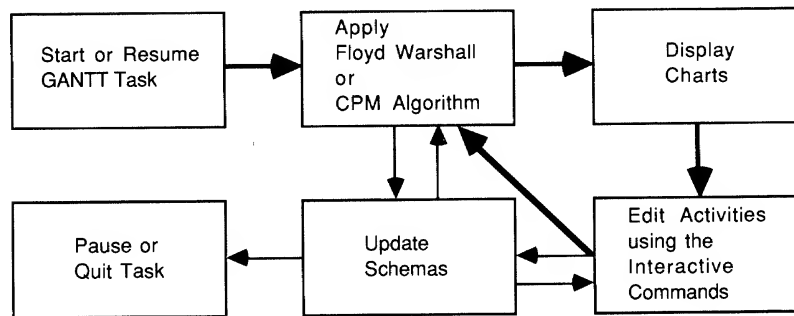


Figure 4-17. Interactive Scheduling Process

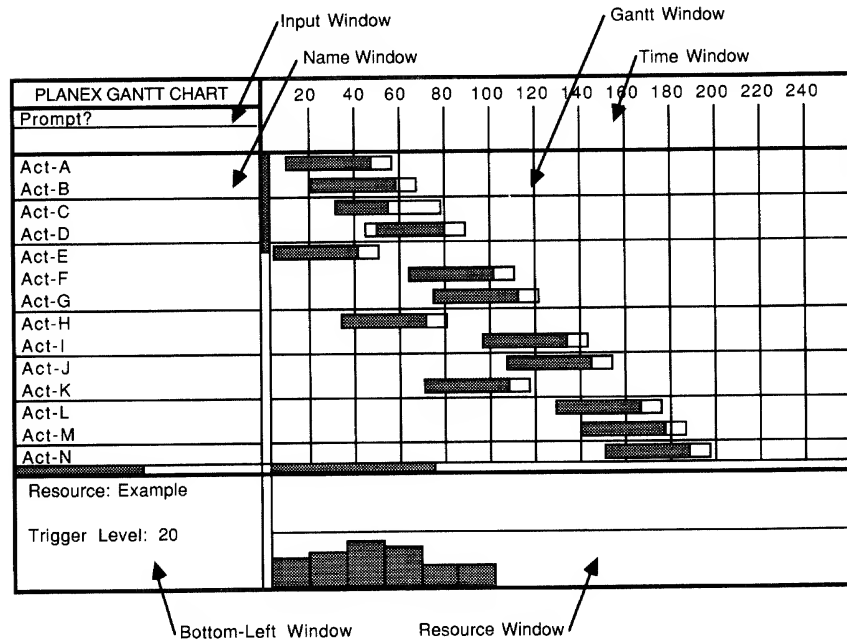


Figure 4-18. Screen Display of GANTT

GANTT has been used as the interactive scheduling component of the CONSTRUCTION PLANEX and EXCAVATION PLANEX systems. Its implementation assumes that the project activity network is represented using the unified model described in Section 3.3. GANTT can be used with any PLANEX application which generates such a project representation.

In addition, an *Input/Output* window used to input commands or edit cells and a *Message* window used to output messages appear at the bottom of the display screen.

The CP provides the means to display and modify the control information stored in the *agenda*. The user may insert goals or changes in the *agenda*, modify the order in which operators are listed, insert domain operators in the queue, or modify operator precedences. In addition, the user may execute any of the control operators of the system to create strategic plans of domain operators.

4.4.4 Report Generator

The REPORT GENERATOR (RG) lets the user design output reports. Each report is described using a *report-format* schema that specifies the columns of the report, the order in which these columns are printed, and any operations applied to the values before they are printed. A report format may also specify constraints on the values included in the report. Individual *column-format* schemas describe the columns in a report. Column formats specify the source of the data and attributes such as column output width or number of decimal places. A column format schema may be used by more than one report format. For example, in CONSTRUCTION PLANEX, a single column format schema for activity names is used in all reports related to project activities.

The use of the RG is illustrated with an example. Assume the planner wants to produce a report of activity costs similar to that of Figure 4-20. The report contains three columns to display different activity attributes. The values in the first two columns ("name" and "cost") are obtained from the activity schemas. However, the values in the third column are not stored in these frames. The "percent" cost of an activity is computed by dividing its "cost" by the total cost of the project. The total project cost is not stored in the context, but is computed by adding all of the activity costs. Furthermore, the user wants the report: (1) to be sorted in ascending order by activity name; and (2) to include only those activities whose percent cost is greater than or equal to 10% of the total project cost.

ACTIVITY COSTS		
Name	Cost	Percent
Activity-1	200.00	10.0
Activity-5	400.00	20.0
Activity-8	300.00	15.0

Figure 4-20. Example of a Report

```
(defschema act-cost-per
  (is-a          report-format)
  (report-title  "ACTIVITY COSTS")
  (format-columns column-name column-cost
                  column-percent)
  (sort-order    1)
  (sort-type     a)
  (solving-order 1 2 3)
  (printing-order 1 2 3)
  (functions-columns (<total> (add-list <cost>)))
  (functions-rows   (<percent> (/ <cost> <total>)))
  (constraints-printing (>= <percent> 10)))
```

Figure 4-21. Report Format Schema for the Report of Figure 4-20

Figure 4-21 shows the report format schema that produces the cost report of Figure 4-20. The slots in this schema are:

- *is-a* identifies the frame as a report format schema.
- *report-title* contains the title of the report.
- *format-columns* specifies the names of the column format schemas included in the report. The example report is composed of three columns corresponding to: (1) the activity name; (2) its cost; and (3) its percent cost.
- *sort-order* indicates how to sort the data. In the example, activities will be sorted on the first column format only (the activity name).
- *sort-type* specifies if the sort is in ascending ("a") or descending ("d") order.
- *solving-order* indicates the sequence in which values are obtained from context objects. This order is important in substituting binding variables in the formulas of the other slots. When producing the desired report, the RG will first obtain the name, then the cost and finally it will compute the percent cost.
- *printing-order* indicates the order of the columns in the report. This order need not correspond to the order in which column formats were listed in the *format-columns* slot.
- *functions-columns* contains a list of (*<binding-variable>* *function*) pairs defining operations to be applied to each column of the report. Before this slot is evaluated, the RG performs the bindings indicated in the column format schemas. For example, before evaluating the function *add-list*, *<cost>* is bound to a list of all the activity costs. This function is evaluated and the result is bound to the variable *<total>*.
- *functions-rows* contains a list of (*<binding-variable>* *function*) pairs defining operations applied to each row of the report. When printing a row, the RG updates the bindings specified in the column format schemas and evaluates the functions sequentially. In the example, the variable *<percent>* is computed as the value of the variable *<cost>* divided by the value of the variable *<total>*.

```
(defschema column-format-cost
  (is-a      column-format)
  (column-title "Cost")
  (from-schema current-object)
  (from-slot   cost)
  (binding-value <cost>)
  (result-type  real)
  (width       8)
  (decimals    2))
```

Figure 4-22. Example of a Column Format Schema

- *constraints-printing* specifies restrictions on the values of variables included in the report. If any of these constraints is not satisfied, the row is omitted from the final report. In the example, only those rows with a *(percent)* greater than or equal to ten (10) are printed.

Figure 4-22 shows the column format schema used by the RG to display a column of activity costs. The slots in this schema are:

- *is-a* identifies the frame as a column format schema.
- *column-title* contains the title of the column.
- *from-schema* indicates the object from which an attribute value will be retrieved. In the example, the *current-object* refers to a particular activity.
- *from-slot* indicates the attribute of the object that will be used to fill the column.
- *binding-value* specifies the binding variable associated with the values in this column.
- *result-type* specifies the data type printed in the column.
- *width* specifies the total width of the column.
- *decimals* defines the number of decimal places used to format numeric output.

The RG has been used to produce many different types of reports in the CONSTRUCTION PLANEX and HARNESS PLANEX systems, such as scheduling, cost and technology reports. In these applications, report format and column format schemas provide all the information the RG needs to generate the desired reports.

4.4.5 Other Interaction Mechanisms

The PLANEX architecture supports several other user interaction mechanisms. These include:

- *Questions to the User* that let the user control the operation of the system or modify attribute values of context objects;

- *Menus* that invoke the execution of domain and control operators;
- *Explanations* that provide information about results and decisions made by the system; and
- *Passive Output Graphics* that display context information in terms of pictures, graphs and charts.

Users may influence the planning process by answering the questions displayed by the system. Consider the interrogative during the execution of the CONSTRUCTION PLANEX operator that determines activity durations, shown in Figure 4-23 (user input is underlined). When the user modifies the number of crews, several values in the activity schema are changed. Similarly, several values are changed when the user decides to use overtime and eliminate fractional work days.

```
***** Duration Information for PA
EXCAVATION-FOUNDATION-P01-S00-B00-F00
```

```
- Crew          CREW-EXCAVATION-05
- Components of Crew  ((1 BACKHOE-3/4)
                      (1 OPERATOR-BACKHOE-3/4))
- Number of Crews    1.04
- Number of days     10
```

Would you like to change any of these settings ? [n] y

Type of change change-crews

-> Give me the number of crews 1.0

```
***** Duration Information for PA
EXCAVATION-FOUNDATION-P01-S00-B00-F00
```

```
- Crew          CREW-EXCAVATION-05
- Components of Crew  ((1 BACKHOE-3/4)
                      (1 OPERATOR-BACKHOE-3/4))
- Number of Crews    1.0
- Number of days     10.42
```

Would you like to change any of these settings ? [n] n

```
**** The duration of PA P01-S00-B00-F00-PA-10-60 is 83.33 hours
**** or 10.42 days
```

Would you like to use overtime in order to eliminate day fractions? y

Figure 4-23. Questions to the User During the Determination of Activity Durations

In the current version of PLANEX, questions are embedded in the procedural codes of the domain operators. There is no generic mechanism to provide this type of interaction, but it might be possible to create such a mechanism.

Menus provide a means to control the execution of domain and control operators. A complete menu-driven interface is composed of many interrelated menus. An example of such an interface for CONSTRUCTION PLANEX is described in Section 6.2.4.

PLANEX provides limited explanations of task-specific operations by storing the names of the KSs used to obtain planning results. When the user requests information describing the manner in which the system computed a particular value, PLANEX displays the name of the KS associated with this object attribute. Suppose the "amount-of-work" of a particular activity object is computed using the KS *KS-Amount-Example*. During the execution of the domain operator, PLANEX creates a slot called *WHY-amount-of-work* to store the value "KS-Amount-Example" for later reference. When the user wants to know *why* the activity has a particular "amount-of-work" value, "KS-Amount-Example" is displayed. A second type of explanation occurs when the user asks *how* a result may be computed. In this case, no inverse pointers exist because the domain operator has not yet been executed. To answer these types of questions, PLANEX finds those operators whose DOSs indicate they produce the requested result and displays their names.

The last type of mechanism used in PLANEX is a set of passive output displays. Pictures are powerful means for displaying data and data relationships. Depending on the application domain, a number of output programs may be added to the system. An example of one such program is the ANIMATOR system described in Chapter 6. ANIMATOR produces construction simulations by displaying some of the project activity information produced by CONSTRUCTION PLANEX. With the ANIMATOR, the user can view how frame buildings are constructed. This program has proven invaluable for detecting errors in activity precedence computations.

4.5 Conclusions

The basic contributions of the PLANEX architecture can be summarized by considering the key points of the architecture, in terms of how PLANEX operates and the capabilities it provides. PLANEX is a system which:

- provides a domain-independent knowledge representation scheme that is independent of the set of domain operators of a particular application;
- supports modular development because domain operators may be added or removed from the system without having to alter the other operators;
- creates and executes strategic meta-plans of domain operators using its control operators;

- distinguishes between the feasibility and desirability of the domain operators by using declarative knowledge;
- provides the user with tools for creating and modifying the knowledge used in an application; and
- provides the user with an interface to control the execution of the system.

Combining these concepts into a single operational system adequate for solving process planning problems is a contribution to the development of process planning systems. In addition, how PLANEX develops a process plan distinguishes it from other planning systems, particularly those described in Chapter 2. These different approaches to problem solving are also contributions of the architecture:

- PLANEX generates not just sequences of actions, but complete *process plans*. The plans include the activities required to construct or manufacture the object; their precedence relationships; the technologies which will be used for performing these activities; the estimated activity durations and costs; and schedule information such as earliest and latest event times.
- PLANEX explicitly generates both *process plans* and *operator plans*. A process plan indicates the manner in which the desired product will be manufactured or constructed. The operator plan is the internal set of problem-solving actions to be performed to produce the process plan. This plan is generated by the control operators and is not related to the product design. Most other systems do not distinguish between these types of plans. Means-end planners do not generate plans about the operators required to generate plans. Meta-planners distinguish between plan actions and *meta-actions* used to generate the plan, but do not strategically plan the execution of the meta-actions. Blackboard planners can produce both types of plans, but do not include explicit operators for strategic planning. In contrast, PLANEX provides a set of control operators to generate operator plans.



5 Developing Process Planning Systems

The PLANEX architecture presented in the previous chapter provides tools to represent and use domain knowledge in solving process planning problems. This chapter describes a procedure for developing process planning systems using the components of PLANEX, and illustrates the use of the architecture with examples from three prototype process planning systems:

- CONSTRUCTION PLANEX generates plans for the excavation and erection of concrete or steel-frame buildings;
- EXCAVATION PLANEX formulates plans for construction site excavation;
- HARNESS PLANEX creates plans for manufacture of automotive electrical harnesses.

In addition, the chapter describes a prototype system that solves *blocks-world* problems using the control components of PLANEX. Blocks-world problems involve moving and stacking blocks and represent a simplified domain for testing automated planning systems. The application of this prototype to problems similar to those presented in Section 2.1 is discussed.

The chapter begins with some simple examples that illustrate the capabilities of PLANEX. These examples show how the architecture is used to produce problem-solving behavior similar to that of other AI-based planning systems. A general procedure for implementing process planning systems using the PLANEX system architecture follows. This procedure is illustrated with examples from the four prototype process planning systems described above. The chapter ends with an evaluation of the PLANEX architecture with respect to the requirements presented in Section 3.3. This evaluation is based on the prototype applications and behavioral capabilities of PLANEX.

5.1 Capabilities of the PLANEX Architecture

5.1.1 Knowledge Representation

The combination of rules grouped into *Knowledge Sources* (KSs) and the KNOWLEDGE SOURCE EVALUATOR (KSE) that processes these rules provides PLANEX with flexible tools for representing domain knowledge. The features include:

- *knowledge abstraction* that represents domain knowledge at different levels of detail;
- *knowledge hierarchies* that represent domain knowledge at different levels of importance; and
- *variable firing mechanisms* that permit KSs to be evaluated differently.

Each of these features is illustrated below with examples from the construction project planning domain.

5.1.1.1 Knowledge Abstraction Consider the problem of determining the successors of formwork construction activities such as *formwork-floor-1*, *formwork-floor-2* and *formwork-floor-3* (e.g., formwork for each floor⁸). Formwork is used to support fresh concrete until it sets. Formwork construction is followed by concrete placement. The system should apply this abstract precedence (*formwork* followed by *pour-concrete*) to each of the specific formwork activities in order to produce specific successors such as *pour-concrete-floor-1*. A single abstract rule of the form:

IF	the type of the activity is formwork
THEN	the type of a successor activity is pour-concrete

is not useful for our purposes as it is not related to specific formwork locations. On the other hand, many specific rules of the form:

IF	the activity is formwork-floor-1
THEN	a successor activity is pour-concrete-floor-1

would produce an unnecessarily large knowledge base.

Figure 5-1 illustrates the problem. A generic operator is to be applied to several objects of the context that share common properties (e.g., the type of the activity) but that differ in specific properties (e.g., the floor where the activity is performed). A generic KS whose conditions are expressed in terms of the common properties is used for this operator. The results returned by this KS contain a portion that is generic for any floor (e.g., *pour-concrete* is a successor

⁸ Floor refers to the location of the activity, not the actual floor slab.

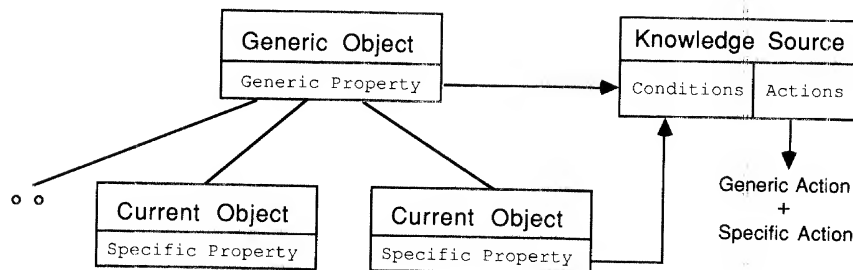


Figure 5-1. Using Abstract Knowledge for Specific Operations

```

(defschema ks-successors-formwork
;
; This ks finds successors for formwork activities
;
  (is-a      ks)
  (ks-type   all)
  (cond-objects current-object current-object)
  (conditions (= type-activity formwork)
               (= floor <floor>))
  (lhs-rules  (T T))
  (rhs-rules  (X X))
  (actions    pour-concrete-floor-<floor>
               remove-forms-floor-<floor>))

```

Figure 5-2. KS that Combines Knowledge at Different Levels of Abstraction

of *formwork*) and a part that is specific to each location (e.g., *pour-concrete* in *floor-1* is a successor of *formwork* in *floor-1*). In this KS, specific attributes of the object to which the operator is applied (e.g., the value of its *floor* attribute) are used to produce the desired results.

Figure 5-2 shows the schema representation of a KS used to find the successors of formwork activities. This KS contains only one rule which checks that the type of activity is *formwork* and binds the variable *<floor>* to the value of the slot *floor* of the activity schema. As described in Section 4.2.3, the KSE interprets everything delimited by “< >” as a binding variable. The value of *<floor>* is then substituted into the generic actions *pour-concrete-floor-<floor>* and *remove-forms-floor-<floor>* to produce the specific actions for each activity (e.g., *pour-concrete-floor-1* and *remove-forms-floor-1*).

5.1.1.2 Knowledge Hierarchies A useful strategy when structuring knowledge bases is to group rules according to relative importance to improve the efficiency of the system. Rules at the top level of a knowledge hierarchy are considered more important than rules in the bottom level. With this structure, and by processing only the important rules first, the search space is reduced.

```

(defschema ks-technology-example
;
; This ks selects appropriate equipment for an excavation
; activity
;
  (is-a      ks)
  (ks-type   first)
  (cond-objects current-object soil-characteristics
               soil-characteristics)
  (conditions (= type-activity excavation)
              (= water-content wet)
              (member type-soil '(soil-1 soil-2)))
  (lhs-rules (T T T)
             (T F I)
             (T T F))
  (rhs-rules (X I I)
             (I X I)
             (I I X))
  (actions   ks-dragline
             ks-clamshell
             ks-power-shovel))

```

Figure 5-3. KS Illustrating Knowledge Hierarchies

```

(defschema ks-duration-example
;
; This ks returns either the expected duration of an activity or
; a three point estimate
;
  (is-a      ks)
  (ks-type   first)
  (cond-objects current-object)
  (conditions (= type-activity excavation))
  (lhs-rules (T)
             (T))
  (rhs-rules (X I I)
             (I X I)
             (I I X))
  (actions   10 20 7))

```

Figure 5-4. KS Illustrating Variable Firing Mechanisms

In PLANEX, knowledge hierarchies are created by structuring KS schemas in the context. Evaluating a KS may result in the evaluation of other KSs that are linked below the first. For example, assume that a domain operator is responsible for selecting appropriate excavation equipment for a certain activity. There are several possible types of equipment (such as power shovels, clamshells or draglines) and within each type of equipment there are many possible subtypes

(e.g., 1/4-cyd power shovel, 1/2-cyd power shovel). One possible knowledge representation is to combine all possible equipment subtypes and all possible factors affecting the choice into a single large KS. However, the decision process may be decomposed into two decisions: (1) select the equipment type; and (2) select the equipment subtype. Such a KS is shown in Figure 5-3. Evaluating this top-level KS leads to the successive evaluation of lower-level KSs such as *KS-dragline*.

5.1.1.3 Variable Firing Mechanisms In PLANEX, rules in a KS are evaluated *sequentially* (in the order written) by the KNOWLEDGE SOURCE EVALUATOR (KSE). Each KS specifies the type of sequential firing used (i.e., if *all* rules are fired or only the *first* applicable one). The firing order may be dynamically changed before evaluating a KS. Consider the KS of Figure 5-4. This KS may return one or three values for activity durations. If the *ks-type* is set equal to "first", only the *expected duration* of the activity is computed. This could be used in deterministic scheduling procedures such as the Basic Critical Path Method that only require point estimates of activity durations to compute the earliest completion time of the project. If the *ks-type* is set equal to "all", then three values are computed. These values correspond to the *expected*, *pessimistic* and *optimistic* activity durations required by some probabilistic scheduling procedures such as the PERT. The example illustrates that the same KS may be used for different purposes by changing how it is evaluated.

5.1.2 Problem Solving and Control

The architecture of PLANEX provides the user with tools to implement different problem-solving behaviors including:

- *hierarchical planning* in which problem-solving operators are represented at different levels of abstraction;
- *nonlinear planning* in which operators are partially ordered by analyzing goals in parallel;
- *meta-planning* in which operators are arranged into layers and operators of one layer control the execution of operators in the layer immediately below; and
- *opportunistic planning* in which operators are applied independently whenever their execution is relevant in solving part of the problem.

This section illustrates the capabilities of the architecture to support these problem-solving behaviors with examples from several domains.

5.1.2.1 Hierarchical Planning In hierarchical planning, solutions are obtained at different levels of abstraction. This requires representing the problem domain at different levels of detail. Upper (abstract) levels in the hierarchy include only those characteristics of the problem considered to be critical (i.e., an abstract representation). Lower levels incorporate the additional characteristics ignored in the upper levels. With this approach, the effort in searching for a solution is reduced. The problem solver searches first for a solution in the upper-level abstraction spaces and uses this solution as a starting point in the lower-level abstraction spaces.

Incorporating hierarchical problem solving in planning systems appears in ABSTRIPS (see p. 20). By classifying operator preconditions with respect to their *criticality*, an *operator hierarchy* is created. A precondition that cannot be changed by an action is considered more critical than others. In upper-level abstraction spaces, preconditions with low criticality are ignored and abstract plans are generated which satisfy only the most critical preconditions. Once a successful plan is generated in a particular abstraction level, ABSTRIPS identifies which previously ignored preconditions are unmet in the abstract plan. Then the system tries to satisfy these less critical preconditions by planning at a more detailed level.

In PLANEX, operator hierarchies similar to those of ABSTRIPS may be implemented by representing each problem-solving operator with different domain operator schemas as shown in Figure 5-5 (only relevant slots are included in this figure). In this example, conditions are slots in an auxiliary object called the *world*. Operator A is described by either of the *Domain Operator Schemas* (DOSs), *DOS-oper-A-1* or *DOS-oper-A-2*. The description in schema *DOS-oper-A-1* includes the important precondition *condition-1* and ignores the less important precondition *condition-2*. When generating an abstract plan, the control operators would use this schema. Schema *DOS-oper-A-2* gives a more detailed description of the operator because both preconditions are included. (The second DOS inherits *Condition-1* via the *is-a* link to the first DOS.) *DOS-operator-A-2* would be used when generating a detailed plan.

An example of hierarchical planning is illustrated using the operator hierarchy of Figure 5-6. Operators are described at three levels of abstraction using DOSs similar to those of Figure 5-5. Suppose that PLANEX is requested to find a plan to achieve *condition-H*. Application of the *Backward Search Operator* (BSO) would produce the following plans:

1. At the most abstract level (level 1), the system identifies operator *H* as needed to achieve *condition-H* and creates a subgoal to satisfy its only precondition *condition-D* (preconditions *condition-G* and *condition-F* are ignored at this level of abstraction). *Condition-D* can be achieved by operator A which has no preconditions. The successful plan is shown in level 1 of Figure 5-7 and contains only the initial goal *condition-H* and the two subgoals, *condition-D* and *condition-A*.

```

(defschema DOS-oper-A-1
  (is-a          operator)
  (input-objects world)
  (input-slots   condition-1)
  (input-bindings nil)
  (input-cond-types filled))

(defschema DOS-oper-A-2
  (is-a          DOS-oper-A-1)
  (input-objects world)
  (input-slots   condition-2)
  (input-bindings nil)
  (input-cond-types filled))

```

Figure 5-5. Alternative Representations of a Domain Operator

2. The system details the level 1 plan at the second abstraction level (level 2) by using the corresponding DOS. The system finds that operator *H* has the unsatisfied precondition *condition-G* and tries to achieve it by starting with the abstract plan as a partial solution. Previously achieved conditions (*condition-A*, *condition-D* and *condition-H*) are considered satisfied and the system does not expand them. The resulting plan is shown in level 2 of Figure 5-7. Applying operator *G* satisfies *condition-G*. Its precondition, *condition-C*, is satisfied by applying operator *C*; precondition *condition-A* for operator *C* is already satisfied.
3. The system details the plan at the lowest abstraction level (level 3). This process introduces the subgoal *condition-F* required by operator *H*. Achieving this subgoal can be accomplished by applying operator *F*, *F'* or *F''*. The system selects operator *F* because its only precondition (*condition-D*) has been satisfied at abstraction level 2. The complete plan is shown in level 3 of Figure 5-7.

If operators had been described at the most detailed level only, the problem-solving process would have been more complex. Proceeding in a breadth-first manner, the BSO would have explored the possibility of satisfying *condition-F* by applying operator *F*, *F'* and *F''* and by expanding their preconditions (*condition-I* and *condition-J*). In the hierarchical planning process, this part of the search space is never explored.

Using operator hierarchies requires a mechanism to identify which DOS corresponds to a particular operator at each level of abstraction. Auxiliary *knowledge sources* (KSs) could be used to establish this correspondence. Evaluation of these KSs would reveal which of the alternative DOSs should be used by the control operators in the planning phase.

<i>Operator</i>	<i>Preconditions @ Level 1</i>	<i>Preconditions @ Level 2</i>	<i>Preconditions @ Level 3</i>	<i>Effect</i>
A	none	none	none	condition-A
B	condition-A	condition-A	condition-A	condition-B
C	condition-A	condition-A	condition-A	condition-C
D	condition-A	condition-A	condition-A	condition-D
E	condition-B	condition-B	condition-B	condition-E
F	condition-D	condition-D	condition-D	condition-F
F'	condition-I	condition-I	condition-I	condition-F
F''	condition-J	condition-J	condition-J	condition-F
G	condition-C	condition-C	condition-C condition-E	condition-G
H	condition-D	condition-D condition-G	condition-D condition-G condition-F	condition-H
I	condition-J	condition-J	condition-J	condition-I
J	condition-K	condition-K	condition-K	condition-J
K	none	none	none	condition-K

Figure 5-6. Example of an Operator Hierarchy

5.1.2.2 Nonlinear Planning PLANEX can be designated as a nonlinear planner by examining two characteristics:

- the type of operator plans produced by the system; and
- the manner in which these plans are generated.

The first characteristic is related to the plan structure and operator execution order. In the planning phase, PLANEX creates *networks* of operators from which different operator sequences may be extracted. The operators are only *partially* ordered in the network because no commitment has been made to specify a unique linear order for operator execution. Assume the system created the network of operators shown at the left in Figure 5-8. Two alternative operator execution sequences that do not violate the operator precedences exist as shown at the right in Figure 5-8. In the execution phase, either of these sequences could be selected by incorporating heuristics or asking the user to pick one of the alternatives.

The second characteristic is related to how the system decomposes a problem into smaller subproblems. The algorithms of the *Forward Propagation* and

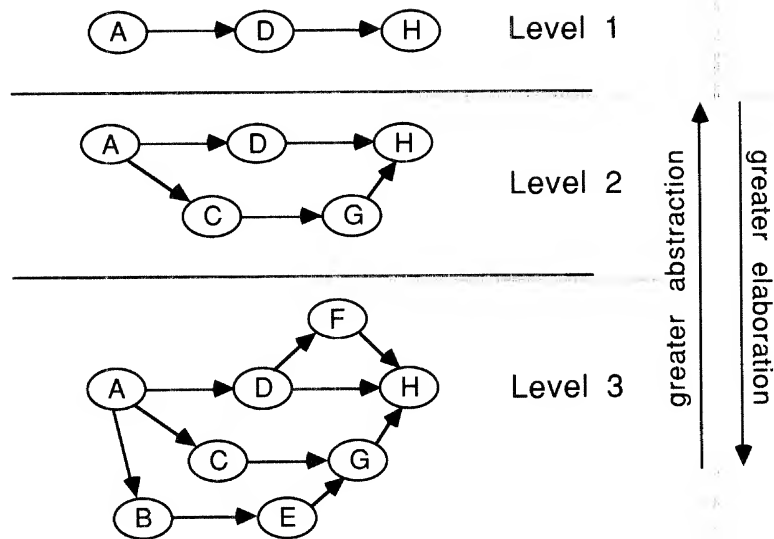


Figure 5-7. Plans Produced in the Hierarchical Planning Example

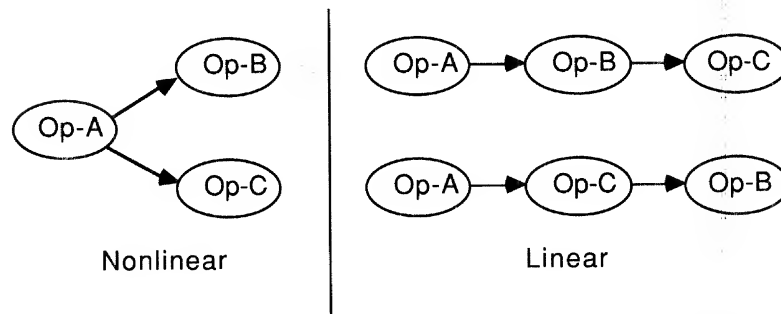


Figure 5-8. Example of a Network of Operators and Operator Sequences

Backward Search control operators generate operator networks by analyzing conditions (context changes or goals) *individually*. The system builds operator sequences for each goal and then merges the sequences to obtain the final operator network. The structure of this network does not depend on the order in which individual goals are considered. In this sense, the planning strategy is similar to that of nonlinear planning systems like NOAH which decomposes the desired *state* of the *world* into its constituent goals and expands them in parallel.

5.1.2.3 Meta-Planning PLANEX has the characteristics of a meta-planner (see p. 30) because:

- its operators are classified as either *domain* or *control* (i.e., two layers); and
- *control* operators determine the execution order of *domain* operators.

The distinction between both types of operators is not merely architectural. PLANEX distinguishes between a *process plan* (e.g., the activity network for a building) and the plan of problem-solving *tasks* required to generate this particular process plan. The outcome of the control planning phase is a *meta-plan* composed of domain operator names.

For some domains, it may be desirable to create additional layers of operators. This can be done by invoking operators from the right-hand-side of the rules in the KS. Suppose that the user wants to create the two-layer operator structure of Figure 5-9. The execution of domain operators *op-A*, *op-B* and *op-C* is controlled by meta-operator *meta-op-A* using the auxiliary KS *KS-meta-op-A*. Such a KS is shown in Figure 5-10. Based on the information stored in context objects *object-x* and *object-y*, this KS may invoke the following sequences of operators:

1. *op-A* \rightarrow *op-C*: the first condition is true and the second condition is false.
2. *op-B* \rightarrow *op-C*: both conditions are false.
3. *op-C*: only the second condition is true.

5.1.2.4 Opportunistic Planning The fundamental problem-solving behavior of PLANEX incorporates both strategic and opportunistic elements. Strategic planning is performed in the *planning* phase when the system creates a plan of problem-solving operators to be executed that will propagate context changes or satisfy goals. To implement purely opportunistic problem solving, this phase could be eliminated and control KSs would select operators from the agenda for execution. The system would repeatedly execute the following *problem-solving cycle*:

- Step 1. Update the agenda.* Those operators with satisfied preconditions are inserted in the agenda.
- Step 2. Choose an operator in the agenda to be executed.* The control KS selects a problem-solving operator from the agenda which is likely to contribute to the solution of the problem.
- Step 3. Execute the selected operator.* The selected operator is executed and the corresponding context changes are recorded.

The second step of the problem-solving cycle chooses a *feasible* operator by considering how it contributes to solving the problem. Defining a control KS to perform this selection is not easy. Several criteria such as: (1) the type of operator (e.g., activity creation, duration estimation); (2) the characteristics of

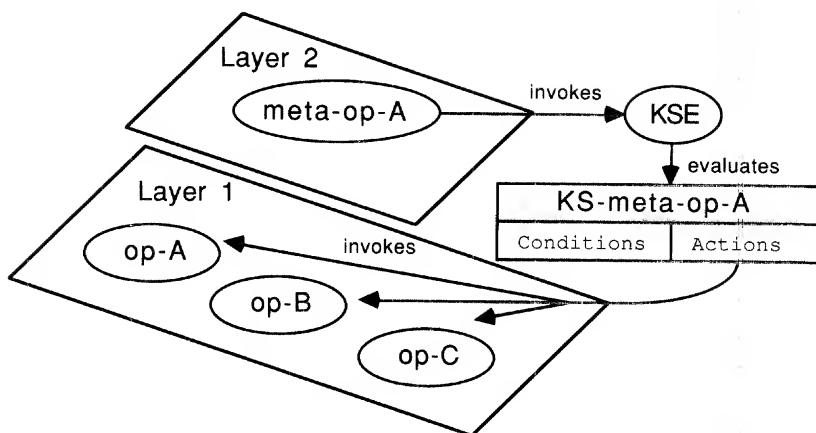


Figure 5-9. Example of a Layered Structure for Operators

```

(defschema ks-meta-op-A
;
; This ks invokes the execution of the domain operators controlled
; by meta operator meta-op-A
;
  (is-a      ks)
  (ks-type   all)
  (cond-objects object-x object-y)
  (conditions (= slot-x value-x)
              (= slot-y value-y))
  (lhs-rules (T F)
             (F F)
             (I I))
  (rhs-rules (X I I)
             (I X I)
             (I I X))
  (actions  (op-A)
            (op-B)
            (op-C)))

```

Figure 5-10. Example of a KS Used by a Meta-Operator

the object to which the operator is applied (e.g., the object represents a concrete pouring activity); and (3) the current state of the problem-solving process (e.g., the cost of activities has not been determined) would have to be incorporated into the control KS to effectively select an operator. Thus, transforming PLANEX into a sophisticated opportunistic planner like OPM would require considerable effort. However, a *primitive* opportunistic problem-solving strategy could be built by considering all operator effects as *unpredictable*. PLANEX would then behave in a forward-chaining manner. The problem-solving cycle

would begin by invoking the *Forward Propagation Operator* (FPO). The FPO would identify the domain operators that are immediately executable based on the information in context objects (Step 1). The system would choose any of these operators (a *primitive* version of Step 2) and then execute it (Step 3). The system would reinvoke the FPO in order to repeat the cycle.

5.2 Development of Process Planning Systems Using PLANEX

This section describes a generic procedure for developing a process planning system using the components of the PLANEX system architecture. This procedure is divided into three major stages:

- *Conceptualization* during which models for process planning operations are developed and types of knowledge and representational structures are identified;
- *Design* during which planning tasks are decomposed into simple planning operations, knowledge sources are structured into hierarchies, and attributes of objects are detailed;
- *Implementation* during which the procedural codes of the planning operators are defined, and the required knowledge sources and schema definitions are created.

Typically, development proceeds iteratively through the three stages until a satisfactory prototype is obtained. Then the scope of the prototype can be extended by adding and refining the knowledge base.

5.2.1 Conceptualization

Conceptualization requires the developer to formulate models for the major planning operations. Examples are the models for the formulation of activities, the selection of technologies, the estimation of activity attributes and the preparation of process schedules. The character of and relationships among these models will affect the types of knowledge and representational structures required to implement a particular application.

As shown in Figure 5-11, planning models are developed by analyzing the design information available, the planning information desired, the existing process planning procedures and the available planning knowledge. For example, assume that the bottom-up activity formulation model presented in Section 3.2.1 is adopted for generating activity networks for a certain type of product. Questions to be answered during the development process would include:

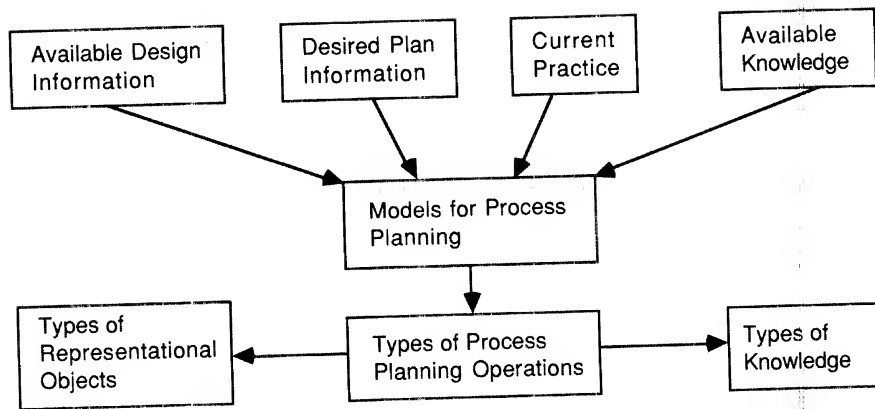


Figure 5-11. Conceptualization of a Process Planning System

- How are activities identified from design drawings and specifications?
- What sources of knowledge (e.g., databases of components, previous drawings, estimating books) are available?
- How is the final product decomposed into design elements?
- What type of element activities are performed to produce components?
- What type of activities should be represented in a process plan?
- How are element activities aggregated into project activities?

Answers to these questions will help to identify the representational structures required for the product and the different levels of activity aggregation in the system.

Planning models for a process planning system may vary across different application domains. Figure 5-12 illustrates three activity formulation and technology selection models for three different process planning systems. Model A is for a system for excavation planning. This model indicates that the selection of general types of excavation equipment precedes activity definition. The model assumes that excavation equipment can be selected without knowledge of which activities are going to use them. Model B (for construction planning) assumes that selection of equipment is made after activities are identified. In this model, activity definition is not dependent upon knowing the technologies which will be used. Model C combines the selection of technologies with the identification of manufacturing activities for a product. This model is more general than models A and B, but may require more knowledge to handle these tasks simultaneously.

Developing and adopting a planning model has implications for the relationships among domain operators, context objects and knowledge sources. For example, defining domain operator schemas and building knowledge sources may be difficult when process planning is decomposed into aggregate, large-

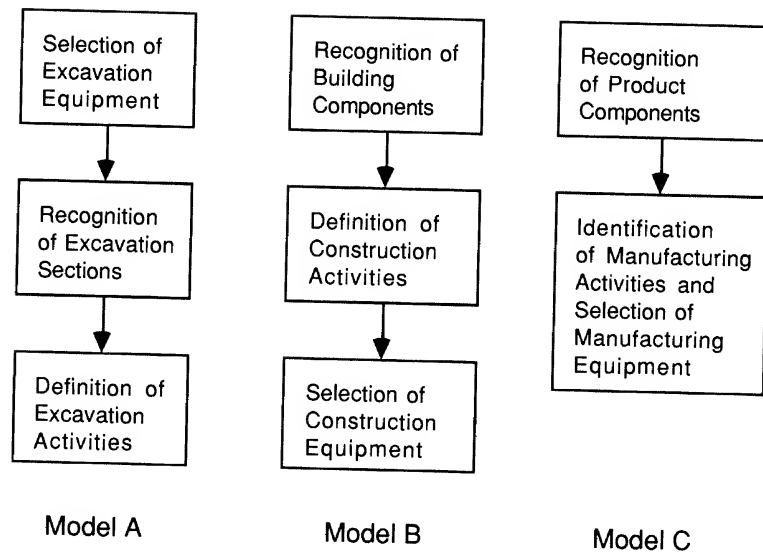


Figure 5-12. Example of Activity Formulation and Technology Choice Models

scale planning operations, whereas a fine-grained decomposition of process planning would require simple procedural codes, small knowledge sources and well-defined context objects. However, some applications may lack flexibility when the planning process is decomposed into very simple operations. This trade-off between simplicity and flexibility can only be resolved by analyzing the characteristics of the particular application.

5.2.2 Design

Design involves detailing the structure of the knowledge sources, domain operators and object types. Figure 5-13 illustrates the tasks needed in designing a process planning system.

In the design stage, the types of knowledge identified in system conceptualization are structured to create knowledge hierarchies. For example, in construction planning, the knowledge related to the selection of materials for activities may be organized using the activity codes of the MASTERFORMAT coding system [18]. This organization facilitates the identification and definition of the knowledge required for particular planning operations.

Another design task which details the information produced in the conceptualization stage is the creation of representational structures. Figure 5-14 shows examples of the representational structures created in the design of a process planning system for excavation tasks. In this application, the holes to be excavated are represented using *hole* objects which are linked to *hole sections*

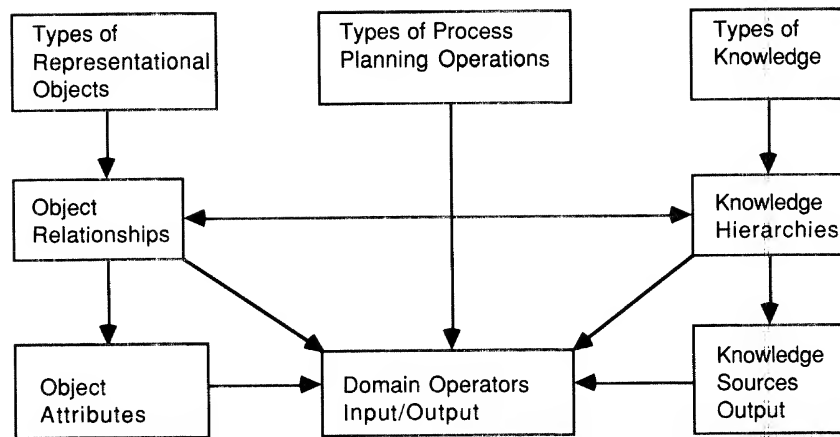


Figure 5-13. Design of a Process Planning System

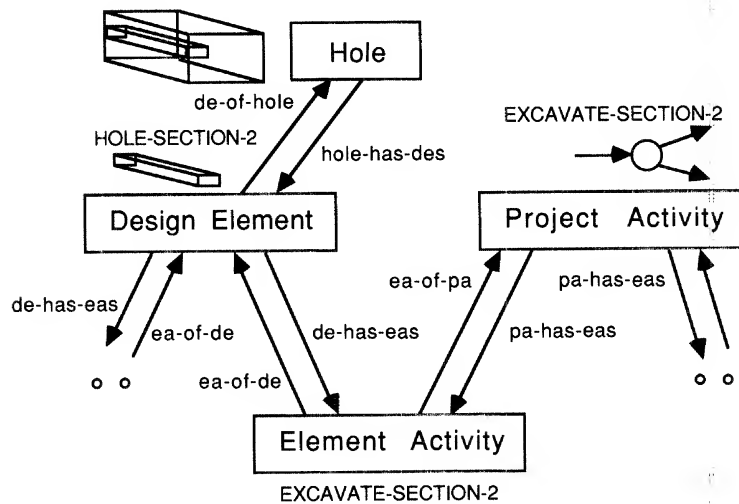


Figure 5-14. Design and Activity Objects of EXCAVATION PLANEX

using *hole-has-des* (hole has design elements) and *de-of-hole* (design element of hole) inverse links. Sections are the basic product components (i.e., the design elements of the bottom-up activity formulation model) from which excavation activities are determined. These activities (i.e., the element activities) are linked to hole sections using *ea-of-de* (element activity of design element) and *de-has-eas* (design element has element activities) links. Finally, element activities are aggregated into project activities using *ea-of-pa* (element activity of

project activity) and *pa-has-eas* (project activity has element activities) relationships.

The creation of representational structures is related to the development of knowledge hierarchies. Thus, in some applications of PLANEX, such as CONSTRUCTION PLANEX, knowledge organization parallels the organization of design elements and activity objects. This case is illustrated in Figure 5-15 (a). However, there are situations in which the knowledge structure is different from the representational structure. Figure 5-15 (b) shows an example of a single knowledge source used for several types of activity objects.

After representational structures and knowledge hierarchies have been created, the names of the slots for each type of context object and the output of each type of knowledge source are specified. Knowledge sources for activity creation in CONSTRUCTION PLANEX return lists of pairs, where the first element of each pair is the name of the activity schema and the second is the name of the activity itself. In contrast, the KSs that compute recommended durations return a single value representing the duration.

The final step of the design stage consists of describing domain operators in terms of their inputs and outputs. In this description, each domain operator is considered to be a "black-box" which retrieves and stores information in the context. Figure 5-16 shows the input/output (I/O) description of the operator that computes the quantity take-offs for element activities in CONSTRUCTION PLANEX. The I/O description also indicates which KS will be evaluated for a particular element activity.

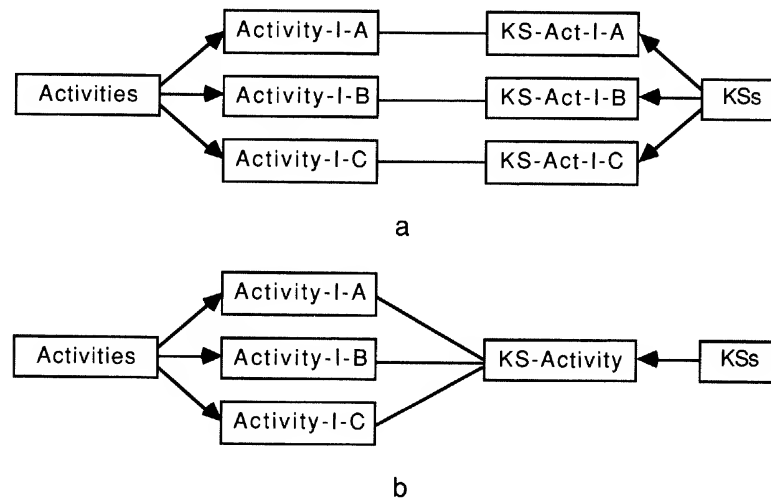


Figure 5-15. Example of Knowledge Hierarchies and Representational Structures

OPERATOR: Compute-Amount-EAS

DESCRIPTION: Computes the work quantities for a list of element activity schemas by evaluating quantity take-off formulas.

INPUT: The geometric dimensions of the design element associated with the activity. In rectangular elements, the slots used are *xl-dimension*, *yl-dimension* and *zl-dimension*.

OUTPUT: The *amount-of-work* of each element activity.

KSSs: KSSs which return the name of the formula which will be evaluated. These KSSs are identified using the *ea-code* value of the element activity schema.

Figure 5-16. Example of Input/Output Description of a Domain Operator

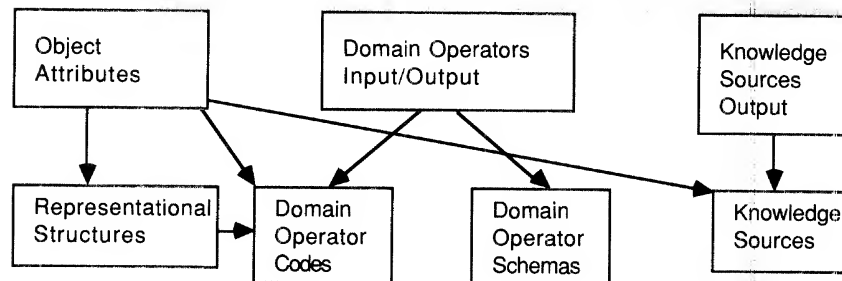


Figure 5-17. Implementation of a Process Planning System

5.2.3 Implementation

Implementation transforms the description of domain operators, objects and KSSs developed during the design stage into instances of PLANEX components. Figure 5-17 shows the four tasks required to implement a system from its design description:

- *Representational Structures* are created from descriptions of object attributes and relationships;
- *Procedural Codes* are written for the domain operators;
- *Domain Operator Schemas* are obtained from I/O descriptions of operators by analyzing the interactions among related operators; and
- *Knowledge Sources* are created for objects of the application domain.

The procedural codes of domain operators are implemented as functions in COMMON LISP. As an example, Figure 5-18 shows the code of the *Determine-Recommended-Duration-PAS* operator of the CONSTRUCTION PLANEX system. The operator is applied to a list of project activity objects. It performs five steps for each object in the list:

```

(defun Determine-Recommended-Duration-PAS (pas)
  (dolist (pa pas t)
    (let* ((pa-code (get-value pa 'pa-code :no-wing t :path nil))
           (ks-name (append-atom 'KS-dura- pa-code)))
      (cond
        ((not (schemap ks-name))
         (format t "~%      ** Cannot determine recommended duration")
         (format t "because this KS has not been loaded > ~s" ks-name)
         nil)
        (t (let ((result (car (evaluate-KS ks-name pa))))
              (cond ((not (numberp result))
                     (format t "~%      ** Recommended Duration of ~s"
                             (format t " is nil" pa)
                             nil)
                     (t (new-value-slot pa 'recommended-duration result)
                        (new-value-slot pa 'why-duration ks-name) result))))))))))

```

Figure 5-18. *Determine-Recommended-Duration* Operator of CONSTRUCTION PLANEX

- Step 1.* Identify the type of activity and the type of design element to which the activity is applied by retrieving the value of the activity *pa-code* slot (e.g., "20-60" for formwork in column footings).
- Step 2.* Create the name of the KS to evaluate by adding the prefix *KS-dura-* to the project activity code (e.g., *KS-dura-20-60*, which is the KS of Figure 5-20).
- Step 3.* If such a KS exists: store the results of its evaluation in the variable *result*; otherwise print an error message and exit.
- Step 4.* If the value of *result* is not a number: print an error message and exit.
- Step 5.* Store the value of *result* in the *recommended-duration* slot of the project activity object. Store the name of the KS in the *why-duration* slot.

Most domain operators of the PLANEX application systems that require KS evaluation are similar to the operator shown in Figure 5-18.

5.3 Example PLANEX Applications

5.3.1 Construction Project Planning

When a contractor decides to bid on a particular project, he has to estimate the duration and cost of the project on the basis of the drawings and specifications. Cost estimates can be obtained by using overall average unit costs or by aggregating the cost of all the building components. For this purpose, the contractor may use a standard cost estimating package. However, estimating the duration or the net-present-value of the project requires formulating a project activity

network and scheduling the project. Currently there are no commercial tools to formulate the activity network. Project planners manually generate suitable activity networks using their experiences with similar projects.

A prototype application of the PLANEX architecture that emphasizes the automated generation of activity networks for construction projects is CONSTRUCTION PLANEX (for a detailed description see Chapter 6). CONSTRUCTION PLANEX uses the expertise of a project planner to generate project activity networks from building design information. The system is intended to perform as an assistant during the formulation of project plans or designs rather than during the evaluation or monitoring of project schedules. The present system plans the excavation and erection of concrete and steel-frame buildings without considering non-structural elements such as partitions, mechanicals or finishes. Knowledge for the system was obtained from the literature and an experienced construction planner [4].

5.3.1.1 Construction Planning Process CONSTRUCTION PLANEX starts with a detailed description of the structural elements of a building (columns, beams, slabs, diagonals and footings) and produces various types of project planning information such as Gantt-charts, project cost curves and scheduling reports. During the planning process, CONSTRUCTION PLANEX also uses information describing the site (e.g., typical characteristics of the soil where the building is located), the contractor (e.g., his minimum attractive rate of return) and some other auxiliary information (e.g., the inflation rate).

The construction planning process proceeds as follows. First, the system analyzes the components of the building and determines the activities required to construct each of these components. Then the system computes the quantity take-off of these activities using the geometry of the building components. Next CONSTRUCTION PLANEX aggregates activities with respect to their type and location, establishes precedences among the activities, and selects technologies to perform the activities. Finally, the system estimates activity attributes such as duration and cost, and prepares a schedule for the project.

5.3.1.2 Representational Structures Figure 5-19 shows the three basic types of context objects used in CONSTRUCTION PLANEX and the names of the links that relate them:

- *design elements* store information about structural components;
- *element activities* represent construction activities required to construct a design element; and
- *project activities* aggregate element activities into more manageable activities for planning purposes.

Design elements may represent single components (e.g., a column) or groups of

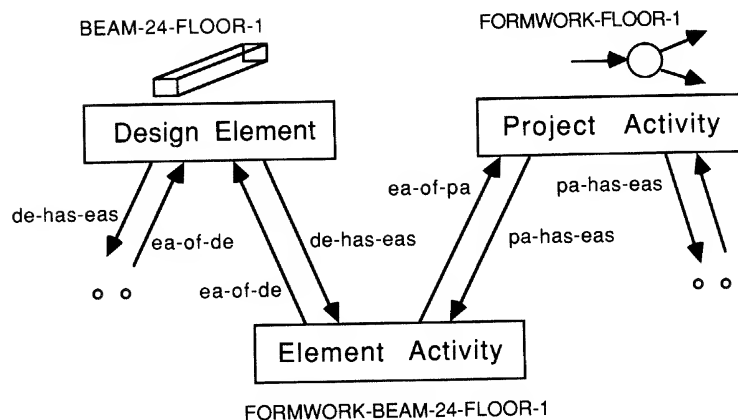


Figure 5-19. Design and Activity Objects Used in CONSTRUCTION PLANEX

components (e.g., a group of concrete columns on the first floor) having the same attributes (e.g., dimensions, material).

5.3.1.3 Knowledge Sources The knowledge base of CONSTRUCTION PLANEX is composed of numerous knowledge sources that provide all the knowledge required during the construction planning process. Examples of these KSs include those storing: the knowledge describing the set of element activities required to construct a design element; the formulas used to compute the quantity take-offs; the manner in which element activities are aggregated; and the appropriate crews for project activities. Each time an operator of CONSTRUCTION PLANEX requires knowledge, one or more KSs are evaluated using the KNOWLEDGE SOURCE EVALUATOR (KSE) and the results of the evaluation are returned to the operator.

A KS of CONSTRUCTION PLANEX is shown in Figure 5-20. This KS contains knowledge describing the recommended duration for placing forms for column footings. This KS is evaluated whenever the operator responsible for computing recommended activity durations needs this knowledge. For example, assume that the system has already created a project activity object for formwork on column footings and that the total area of forms to be placed is 2500 square feet. When the system is determining the recommended duration of this activity, the KS of Figure 5-20 is evaluated and the result of this evaluation (5 days) is stored in a slot of the project activity object.

```

(defschema KS-Dura-20-60
;;
;; This KS indicates the recommended duration for placing forms
;; on column footings. It has three rules:
;; - IF the amount of work is less than or equal to
;;   2,000 sq-ft, THEN 5 days is an appropriate duration.
;; - otherwise, IF the amount of work is less than or equal to
;;   4,000 sq-ft, THEN 10 days is an appropriate duration.
;; - otherwise, IF the amount of work is greater than 4,000 sq-ft,
;;   15 days is an appropriate duration.
;;
  (is-a      ks)
  (ks-name   KS-dura-20-60)
  (ks-type   first)
  (cond-objects current-object
                current-object)
  (conditions (<= amount-of-work-pa 2000)
              (<= amount-of-work-pa 4000))
  (lhs-rules (T I)
              (F T)
              (I F))
  (rhs-rules (X I I)
              (I X I)
              (I I X))
  (actions 5 10 15))

```

Figure 5-20. Example of a Duration KS Used in CONSTRUCTION PLANEX

5.3.1.4 Domain Operators CONSTRUCTION PLANEX generates construction project plans by applying three types of operators:

- *design element operators* act on design element objects. Examples of design element operators are those for grouping design elements and creating element activities.
- *element activity operators* act on element activity objects. Examples of element activity operators are those for computing quantity take-offs and aggregating element activities.
- *project activity operators* act on project activity objects. Examples of project activity operators are those for selecting crews, establishing activity precedences and estimating project activity durations.

An example of a project activity domain operator was shown in Figure 5-18 (*Determine-Recommended-Duration-PAS*).

5.3.1.5 User Interface The user interface of CONSTRUCTION PLANEX contains examples of all the user interaction mechanisms of PLANEX presented in Section 4.4. The system provides an Activity-On-Arrow diagram of the project network, an interactive Gantt-chart and two types of project cost curves.

5.3.2 *Excavation Project Planning*

EXCAVATION PLANEX is knowledge-based planning system which generates project networks for excavation projects [65, 83]. The system recommends appropriate excavation equipment, defines gross vehicle movements, creates a network of excavation activities and estimates the duration of the project.

The EXCAVATION PLANEX system is capable of planning the excavation of rectangular holes with vertical walls and flat bottoms. System knowledge was obtained from books and manuals describing excavation projects.

5.3.2.1 Excavation Planning Process The planning process of EXCAVATION PLANEX proceeds as follows. First, the system selects a general type of equipment for excavation and hauling on the basis of information extracted from the excavation drawings and available terrain information. Then the system divides each hole into excavation sections based upon the particular capabilities of the chosen excavation equipment (e.g., a function of its bucket size). EXCAVATION PLANEX then determines an order for excavating the sections, identifies which activities to execute for each section and structures the activities into a network. Finally, the system estimates activity durations and computes a schedule.

The problem-solving operators and user interface mechanisms of EXCAVATION PLANEX are almost identical to those of CONSTRUCTION PLANEX. However, there are some fundamental differences between the two systems:

- In EXCAVATION PLANEX, the type of excavation and hauling equipment is chosen before project activities are generated. In CONSTRUCTION PLANEX, project activities are created before technologies are selected.
- EXCAVATION PLANEX generates design elements using information describing the type of hole to be excavated and the types of machines to be used. CONSTRUCTION PLANEX starts with the design elements as input.

5.3.2.2 Representational Structures The context objects of EXCAVATION PLANEX are identical to those of CONSTRUCTION PLANEX. The only difference is that EXCAVATION PLANEX decomposes design information into two levels:

- a *hole* level which is input directly from the excavation drawings; and
- a *hole section* level which is generated by the system and represents portions of the hole.

The relationships between the schemas of both levels was shown in Figure 5-14.

Hole sections are similar to the design elements of CONSTRUCTION PLANEX because they represent components of the final product (i.e., the excavation) with an associated set of element activities.

In EXCAVATION PLANEX, each project activity consists of only one element activity. Therefore, the activity network includes explicit excavation activities for every hole section. Although this structure seems appropriate for simple excavation projects (e.g., a single hole with few machines), it will produce extremely large networks for more complex projects. Using an aggregation structure similar to CONSTRUCTION PLANEX, in which a project activity includes several element activities, is a viable solution to this problem.

5.3.2.3 Knowledge Sources The knowledge base of EXCAVATION PLANEX consists of various KSs that resemble those of the CONSTRUCTION PLANEX system. There are similarities with respect to the types of KSs used, the structure of these KSs and the associated domain operators. For example, both systems have *Successor* KSs used by the operator responsible for establishing precedences among project activities. There are some differences, however, in how the systems estimate activity durations. In CONSTRUCTION PLANEX, one or more machines may be used to perform a particular project activity. How many machines to use is determined with heuristics for *recommended* duration, which are based on the total quantity of work (see Figure 5-20). In EXCAVATION PLANEX, each activity corresponds to a hole section and is performed by a single machine. Therefore, the duration is a simple function of the total amount of work, the bucket size and the cycle time of the selected machine.

5.3.2.4 Domain Operators Most of the domain operators used in EXCAVATION PLANEX were taken directly from CONSTRUCTION PLANEX. The scheduling operators were simplified because EXCAVATION PLANEX assumes that all activity precedences are of the *Finish-to-Start* type. Several new operators were required:

- *Select-Machine* chooses either a backhoe or a loader for the excavation;
- *Find-Nearest-Point* identifies the corner of the hole that is closest to the initial position of the machine;
- *Create-Init-Move* creates an object containing information about the distance that machines will have to move before excavation proceeds; and
- *Create-Sections* decomposes the initial hole into design elements representing hole sections.

Figure 5-21 shows the code of the *select-machine* operator. This operator performs five steps:

```

(defun select-machine (hole-info)
  (let* ((machines (car (evaluate-KS 'KS-Select-Excavator
                                   hole-info)))
         (no-machines (length machines))
         (hauler nil) ;; setq'ed below
         (excavator nil))
    (cond ((eq no-machines 1) ;; machine is both hauler
           (setq excavator (car machines)) ;; and excavator
           (setq hauler excavator))
          ((eq no-machines 2) ;; first machine is excavator,
           ;; second is hauler
           (setq excavator (nth 0 machines))
           (setq hauler (nth 1 machines)))
          (t (format t "~%Error in fcn select-machine~%"))))
    (new-value-slot hole-info 'excavator excavator)
    (new-value-slot hole-info 'hauler hauler)
    (let* ((exc-instance (get-value excavator 'instance))
           (haul-instance (get-value hauler 'instance))
           (KS-excavator
            (append-atom-list (list 'KS-
                                   (get-value exc-instance 'is-a)
                                   '-bucket-factor)))
           (KS-hauler
            (append-atom-list (list 'KS-
                                   (get-value haul-instance 'is-a)
                                   '-bucket-factor))))
      (exc-bucket-factor (car (evaluate-KS KS-excavator)))
      (haul-bucket-factor (car (evaluate-KS KS-hauler))))
    (when (equal 'ask-user exc-bucket-factor)
      (format t "~% Unknown bucket factor for excavator ~a~%"
              excavator)
      (format t " in soil ~a. Give me the bucket factor: "
              (get-value 'soil-info 'soil-type))
      (setq exc-bucket-factor (read))
      (format t "~%"))
    (when (equal 'ask-user haul-bucket-factor)
      (format t "~% Unknown bucket factor for hauler ~a~%"
              hauler)
      (format t " in soil ~a. Give me the bucket factor: "
              (get-value 'soil-info 'soil-type))
      (setq haul-bucket-factor (read))
      (format t "~%"))
    (new-value-slot hole-info 'exc-bucket-factor
                     exc-bucket-factor)
    (new-value-slot hole-info 'haul-bucket-factor
                     haul-bucket-factor))))

```

Figure 5-21. *Select-Machine* Operator of EXCAVATION PLANEX

- Step 1.* Evaluate the KS *KS-Select-Excavator*. The result of this evaluation is either: (1) a list with two machine names, one for excavation and one for hauling; or (2) a list with only one machine to be used for both activities.
- Step 2.* Store the names of the excavation and hauling machines in the *hole-info* schema.
- Step 3.* Create the names of the KSs used to select the bucket size (the *bucket factors*).
- Step 4.* Evaluate these KSs to compute the bucket factors.
- Step 5.* Store the bucket factors in the *hole-info* schema.

5.3.2.5 User Interface EXCAVATION PLANEX uses most of the user interface mechanisms of PLANEX. However, the current version of the system has some limitations:

- it does not include an interactive graphical display environment such as GANTT;
- the only output graphics are an Activity-On-Arrow diagram and a simulation of the excavation process; and
- control is provided through a tree of command menus that invokes the control operators of PLANEX.

5.3.3 Manufacturing Process Planning

HARNESS PLANEX is a knowledge-based system that generates activity plans for manufacturing automotive electrical harnesses (described in detail in Chapter 7). The system identifies the manufacturing activities required to produce a particular harness, recommends appropriate equipment to perform these activities and estimates the duration of the manufacturing process. The result of the planning process is a *process sheet* report which is used by the harness manufacturer on the shop floor.

Currently HARNESS PLANEX is capable of planning the manufacture of individual harnesses. It cannot schedule the entire production line. Knowledge in the system is limited to that pertaining to certain harness components and was developed using information provided by an experienced harness manufacturer.

5.3.3.1 Manufacturing Planning Process The planning process of HARNESS PLANEX proceeds as follows. First, the system analyzes information extracted from drawings and builds a model of the harness as a network of context objects. The system aggregates harness wires with common connections into *subassemblies* representing components which can be manufactured independently. HARNESS PLANEX then identifies activities required to cut the wires and

apply terminals to wire ends. The system selects equipment for each manufacturing activity and resolves conflicts among cutting machines so that appropriate machines are assigned to cut each end of the wire. Finally, the system estimates the total time required to produce the harness and the usage level of each machine.

5.3.3.2 Representational Structures The representational structures used in HARNESS PLANEX are more complex than those used in the CONSTRUCTION PLANEX or EXCAVATION PLANEX systems because they model not only individual components of the harness but also the harness topology. HARNESS PLANEX has the following types of objects in its context:

- *wire* objects store the descriptions of the wires which comprise the harness;
- *body* objects represent the central portion of wires;
- *extreme* objects represent the ends of wires;
- *terminal-location* objects represent joints connecting wire ends;
- *activity* objects represent manufacturing activities for wire ends or wire bodies; and
- *machine* objects represent machines used in manufacturing the harness.

The manner in which these types of objects are linked to form representational structures is described in Chapter 7.

5.3.3.3 Knowledge Sources The knowledge base of HARNESS PLANEX is composed of the following types of KSs:

- *Activity* KSs define the set of manufacturing activities for wire body or wire extreme objects;
- *Technology* KSs provide recommendations of appropriate machines for manufacturing activities; and
- *Duration* KSs estimate the expected duration of manufacturing activities.

In addition, the system uses a *Peeling* KS which estimates the length of insulation that should be peeled from a wire end depending upon the type of terminal that will be attached.

5.3.3.4 Domain Operators Using a planning process similar to those used for construction and excavation planning, HARNESS PLANEX generates process sheets by applying simple domain operators which act on context objects. There are operators to: (1) create or delete objects containing harness design information; (2) identify manufacturing activities; (3) select appropriate machines for cutting, tinning or splicing activities; (4) estimate the duration of manufacturing activities; (5) accumulate machine usages; and (6) estimate appropriate peeling lengths of wire ends.

```

(defun select-technology (actv)
  (let* ((act-type (get-value actv 'act-type :no-wing t))
        (ks-name (append-atom 'KS-technology- act-type))
        (machine (cond ((schemap ks-name)
                        (car (evaluate-KS ks-name actv)))
                        (t nil))))
    (cond ((null machine)
           (format t "~% --> No technology for ")
           (format t "activity ~s~% " actv)))
    (new-value-slot actv 'technology machine)
    (add-value-slot machine 'used-by actv)))

```

Figure 5-22. Code of the *Select-Technology* Operator of HARNESS PLANEX

The procedural code of the *Select-Technology* operator is shown in Figure 5-22. Other domain operators have similar codes. When applied to an activity, the *Select-Technology* operator performs the following steps:

- Step 1.* Identify the type of activity (e.g., *cut*).
- Step 2.* Create the name of the KS to be evaluated (e.g., *KS-technology-cut*).
- Step 3.* If the KS exists: store the result of its evaluation in the variable *machine*; otherwise exit.
- Step 4.* If no machine was selected, print an error message.
- Step 5.* Store the value of *machine* in the *technology* slot of the activity object.

The five steps listed above are the same as those performed by the *Determine-Recommended-Duration-PAS* operator of CONSTRUCTION PLANEX (see p. 154). In general, all of the domain operators of the construction, excavation and manufacturing applications perform these five steps.

5.3.3.5 User Interface The system uses some of the user interaction mechanisms of PLANEX described in Section 4.4, such as command menus and questions to the user; however, the interface of HARNESS PLANEX does not include graphical displays. Results of the planning process are presented in various reports produced with the REPORT GENERATOR.

5.3.4 Blocks-World Planning

The previous sections described three similar applications of the PLANEX architecture that generate *process plans* for creating *products*. This section describes a small system, called BLOCKS PLANEX, which solves blocks-world problems using only the control components of PLANEX (i.e., domain KSs are not required). The problem is different from the preceding examples because the output of the system is not a complete *process plan* (i.e., it does not include durations, resources, etc.). However, this application of PLANEX is interesting because it:

- illustrates how the control operators of PLANEX generate strategic plans of domain operators;
- is the basis for a comparison of the control operators of PLANEX with the AI planners reviewed in Section 2.1; and
- shows that BLOCKS PLANEX can solve problems with *double-cross conflicts* [15] without having to order the goals or use complicated critics.

5.3.4.1 Overview of BLOCKS PLANEX BLOCKS PLANEX is implemented with the control components of the PLANEX architecture:

- *Domain Operator Schemas* (DOSs) describe the preconditions and effects of the problem-solving operators;
- the *agenda* stores an operator queue, operator precedences and goals; and
- *control operators* use and modify the information stored in the agenda to generate operator networks.

The description of the blocks-world is stored in *block schemas*. Each block is described by a schema that specifies its location with respect to the other blocks and the table. In this representation, a slot is *filled* whenever a condition is "true" and empty otherwise. This representation permits the control operators of PLANEX to be used directly.

An example of a block schema is shown in Figure 5-23. The schema indicates that block C is above block A and that block A is on the table. The schema has the following slots:

- An *is-a* slot identifies the schema as a block schema.
- A *name* slot indicates the name of the block described by the schema. Each block has a particular *name* slot. For example, block A has a slot titled *name-a* which is filled with its name, and the schema for block B has a *name-b* slot which is filled with "block-b".
- The *has-above* slots indicate which block is on top of the block described by the schema (only one of these slots can be filled at any time in the planning process).
- A *cleartop* slot is filled whenever nothing is on top of the block.
- An *on-table* slot is filled whenever the block is on the table.

```
(defschema      block-a
  (is-a         block)
  (name-a       block-a)
  (has-b-above)
  (has-c-above yes)
  (cleartop)
  (on-table     yes))
```

Figure 5-23. Example of a Block Schema Used in BLOCKS PLANEX

```

(defschema pick-a-from
  (is-a                operator)
  (domain-type         block)
  (input-objects       current-object current-object
                        block-a)
  (input-slots         name-a has-a-above cleartop)
  (input-bindings      nil nil nil)
  (input-cond-types    erased filled filled)
  (output-objects      current-object current-object
                        block-a)
  (output-slots        has-a-above cleartop on-table)
  (output-bindings     nil nil nil)
  (output-predictable  yes yes yes)
  (output-effect-types erase fill fill))

```

Figure 5-24. Example of a *Pick* Operator Schema Used in BLOCKS PLANEX

BLOCKS PLANEX has two types of domain operators to plan the movements that will achieve the desired final positions:

- *pick* operators pick a block from the top of a stack and put it on the table; and
- *put* operators place one block on top of another.

Pick operators are similar to the *clear* operator and *put* operators to the *puton* operators described in Chapter 2. *Pick* operators only move the top block in a stack.

An example of a *domain operator schema* (DOS) for a *pick* operator is shown in Figure 5-24. The schema indicates that to remove block A from block X (the operator is applied to block X), the following conditions must be true:

- the block to which the operator is applied (e.g., block X) must not be block A;
- the block to which the operator is applied must have block A above it; and
- block A must have nothing on top of it.

Similarly, the DOS indicates that the following effects are achieved by applying the operator:

- block A will no longer be above the block to which the operator is applied (e.g., block X);
- the block to which the operator is applied will have no block on top of it (e.g., *cleartop* is "true"); and
- block A will be on the table.

An example of a DOS for a *put* operator is shown in Figure 5-25. The schema indicates that to place block A on top of block X (the operator is applied to block X), the following conditions must be true:

- the block to which the operator is applied (e.g., block X) must not be block A;

```

(defschema put-a-on
  (is-a          operator)
  (domain-type   block)
  (application-object current-object)
  (input-objects current-object current-object
                 block-a block-a)
  (input-slots   name-a cleartop on-table cleartop)
  (input-bindings nil nil nil nil)
  (input-cond-types erased filled filled filled)
  (output-objects current-object current-object
                 block-a)
  (output-slots   cleartop has-a-above on-table)
  (output-bindings nil nil)
  (output-predictable yes yes yes)
  (output-effect-types erase fill erase))

```

Figure 5-25. Example of a *Put* Operator Schema Used in BLOCKS PLANEX

- the block to which the operator is applied must have nothing on top of it;
- block A must have nothing on top of it; and
- block A must be on the table.

The DOS also indicates the following effects of the operator:

- the block to which the operator is applied (e.g., block X) will no longer be the top of the stack (e.g., *cleartop* is “false”);
- the block to which the operator is applied will have block A above it;
- block A will no longer be on the table.

In BLOCKS PLANEX, domain operators are applied to a *single* block object. Therefore, one operator is required for each operator type (*pick* or *put*) for each block in the problem. Thus, in problems with *N* blocks, there are a total of *N pick* and *N put* operators. In the prototype, these $2 \times N$ operators are predefined. However, the schemas could be created from the problem description by another operator at the start of the problem-solving process.

5.3.4.2 Three-Block Example Consider the example problem of Figure 5-26. This problem was used in Section 2.1 to illustrate the behavior of AI-based planning systems such as STRIPS, INTERPLAN and NOAH. Figure 5-27 shows the schemas defining the initial position of the blocks and the goals. The goals in the agenda frame represent the desired state of the world: a stack of blocks (A, B, C) with A on top of the stack and C on the table.

Problem solution requires two steps:

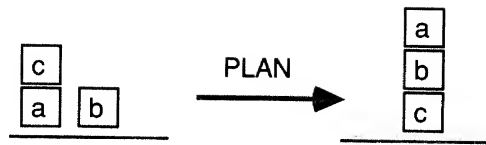


Figure 5-26. Example of a Three-Block Problem

```

(defschema block-a
  (is-a      block)
  (name-a    block-a)
  (has-b-above)
  (has-c-above yes)
  (cleartop)
  (on-table  yes))

(defschema block-b
  (is-a      block)
  (name-b    block-b)
  (has-a-above)
  (has-c-above)
  (cleartop)
  (on-table  yes))

(defschema block-c
  (is-a      block)
  (name-c    block-c)
  (has-a-above)
  (has-b-above)
  (cleartop  yes)
  (on-table))

(defschema agenda
  (context-changes)
  (goals      (block-a cleartop filled)
              (block-b has-a-above filled)
              (block-c has-b-above filled))
  (operator-queue)
  (operator-precedences)
  (operator-preconditions)
  (effect-operators))

```

Figure 5-27. Initial Block Schemas and Agenda for the Three-Block Problem

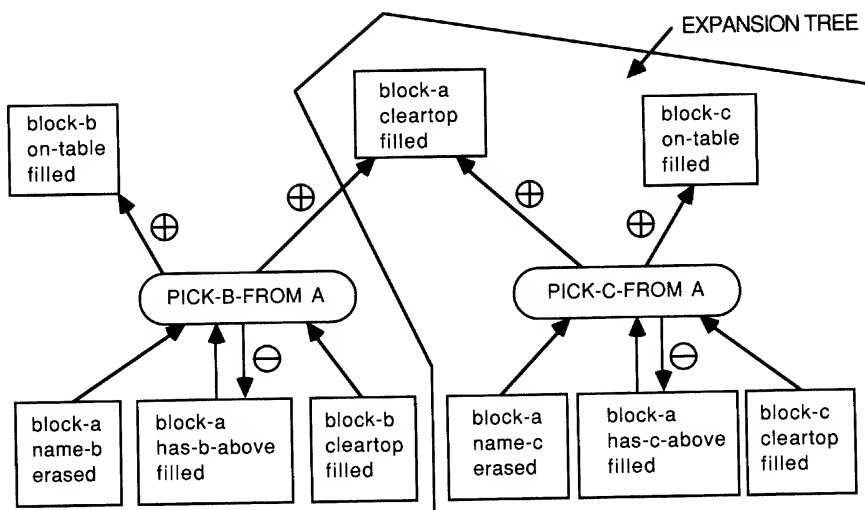


Figure 5-28. Expansion of a Goal in the Three-Block Problem

Step 1. Application of the Backward Search Operator (BSO). The system: (1) searches for a sequence of operators that achieve each goal independently; and (2) merges all sequences into a *global* network of operators and conditions.

Step 2. Application of the Network Interpretation Operator (NIO). The system interprets the global network and formulates an *operator* network.

The application of the BSO proceeds as follows:

- Goal (*block-a cleartop filled*) is expanded as shown in Figure 5-28. There are two possible operators that can achieve the goal: (*pick-b-from a*) and (*pick-c-from a*). Operator (*pick-a-from a*) is not considered because it requires precondition (*block-a cleartop filled*) which is the current goal. All of the preconditions of operator (*pick-c-from a*) are true. Therefore, it is executable and all its effects are labeled achievable.
- Goal (*block-b has-a-above filled*) is expanded. All of its preconditions are true or have been labeled achievable.
- Goal (*block-c has-b-above filled*) is expanded. All of its preconditions are true.

The resulting global network of operators and conditions is shown in Figure 5-29. The *effect-operators* and *operator-preconditions* slots of the agenda store the network.

The application of the NIO to the network of Figure 5-29 yields the operator network of Figure 5-30. The only feasible sequence in this network the desired plan: (*pick-c-from a*) \rightarrow (*put-b-on c*) \rightarrow (*put-a-on b*).

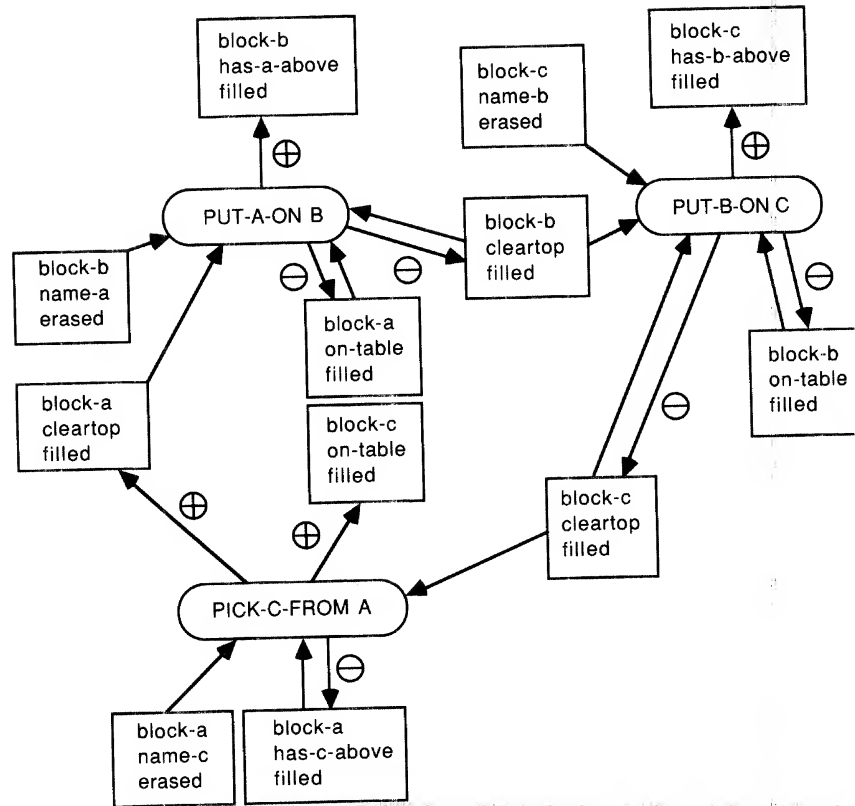


Figure 5-29. Global Network of Operators and Conditions for the Three-Block Problem

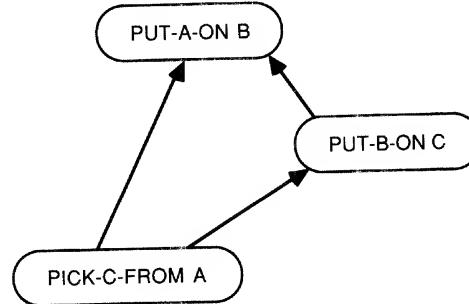


Figure 5-30. Operator Network for the Three-Block Problem

5.3.4.3 Four-Block Example Another example blocks-world problem is shown in Figure 5-31. Corkill refers to this four-block problem as an example with *double cross conflicts* [15]. He describes a solution using NOAH's procedural nets and multiple processors. In his solution, the problem requires two processors that send messages to each other to indicate when they may proceed with the expansion process and when they should stop.

Similarly to the previous example, BLOCKS PLANEX solves the problem by generating a global network of operators and conditions and interpreting this network to identify operator precedences. Figure 5-32 shows the global network of operators and conditions obtained by BLOCKS PLANEX. Interpreting this network yields the operator network of Figure 5-33. There are four possible problem-solving sequences:

(pick-c-from a) → (pick-d-from b) → (put-c-on b) → (put-d-on a)
 (pick-c-from a) → (pick-d-from b) → (put-d-on a) → (put-c-on b)
 (pick-d-from b) → (pick-c-from a) → (put-c-on b) → (put-d-on a)
 (pick-d-from b) → (pick-c-from a) → (put-d-on a) → (put-c-on b)

5.3.4.4 Comments on the Application The examples show that the *Backward Search* and *Network Interpretation Operators* of the PLANEX architecture are applicable in solving blocks-world problems. The solution strategy of BLOCKS PLANEX is different from the procedures followed by other AI planners:

- BLOCKS PLANEX considers operators, preconditions and effects as part of the same network. In planners such as NOAH and NONLIN, preconditions and effects are included in the bodies of the operators.
- In other AI planners, *critics* are used during the expansion process to solve conflicts among actions and avoid redundancies. In BLOCKS PLANEX, critics are not used during the planning process; the system first generates the global network and then extracts the operator network.
- BLOCKS PLANEX expands each goal independently. In contrast, NOAH repeatedly expands a procedural net with all goals.

5.4 Evaluation of the PLANEX Architecture

This section presents a partial evaluation of the PLANEX architecture with respect to the list of requirements presented in Section 3.3. For some requirements, the analysis is supported with examples from the prototype planning systems described in this chapter. For other requirements, however, the analysis is based on the characteristics of the system architecture described in Section 3.3. This distinction is needed because the prototype systems illustrate only some of the capabilities of the knowledge representation and problem-solving tools of PLANEX.



Figure 5-31. Example of a Four-Block Problem

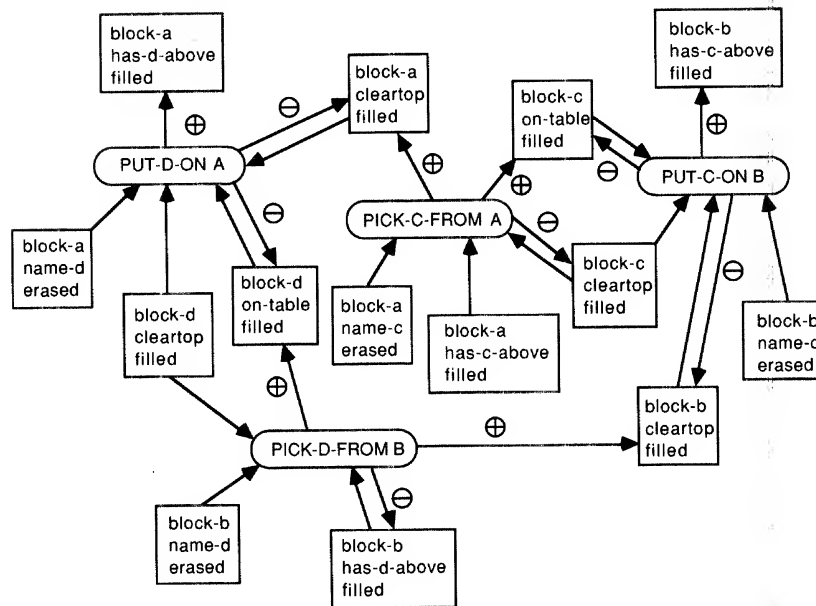


Figure 5-32. Global Network of Operators and Conditions for the Four-Block Problem

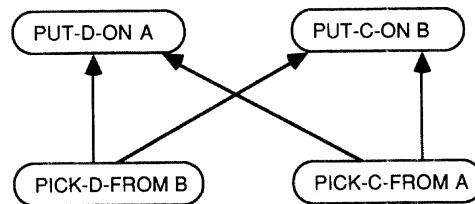


Figure 5-33. Operator Network for the Four-Block Problem

5.4.1 Knowledge Representation

The following requirements for representing process planning knowledge were established. PLANEX should:

1. *Provide a process-independent knowledge representation.* PLANEX meets this requirement by using knowledge represented in *Knowledge Sources* (KSs) that have the generic structure described Section 4.2. This representation is independent of the process planning domain. In the prototype systems for construction, excavation and manufacturing planning described in this chapter, all process planning knowledge was represented in the KS format. Although the KSs used in these systems differ in their content (e.g., the number of rules, the type of values they return), all of them are evaluated using the KNOWLEDGE SOURCE EVALUATOR provided by the architecture.
2. *Provide an operator-independent knowledge representation.* This requirement is also satisfied as the same KS format and syntax was used in all of the prototypes, independent of the type of knowledge represented.
3. *Provide the means to structure knowledge hierarchically.* This architectural requirement is also met because KSs may be structured hierarchically. An example of a KS illustrating knowledge hierarchies is that for the selection of excavation equipment given in Figure 5-3. This is a hypothetical KS as it is not a part of EXCAVATION PLANEX. Knowledge hierarchies were not required in the prototypes as the knowledge could be divided into multiple independent KSs. These KSs were classified with respect to the type of knowledge they represent (e.g., recommended duration for an activity, appropriate technology for a group of activities) and the type of object used for their evaluation (e.g., a formwork activity). This classification of process knowledge made it possible to generate the name of the KS required by the domain operators. Such a horizontal decomposition of the knowledge base may not be desirable or possible in other domains. Thus, it is important for the PLANEX architecture to provide the means to hierarchically structure the knowledge sources.
4. *Provide the means to check the completeness or consistency of an operator's knowledge.* The current version of PLANEX does not include mechanisms to check the consistency of knowledge in a KS. Neither does it provide means for checking the completeness of this knowledge. Therefore, this requirement has not been met in the PLANEX architecture. However, algorithms used for checking the consistency and completeness of decision tables [55, 59, 82] could be adopted to check individual KSs. Incorporating these algorithms into the KNOWLEDGE SOURCE ACQUISITION MODULE would satisfy this requirement at the individual KS level.

5.4.2 Problem-Solving Operators

The following requirements for problem-solving operators were established. PLANEX should:

1. *Achieve operator modularity.* The architecture achieves operator modularity because domain operators are independent. In the prototype systems described, a domain operator does not know about any other domain operator. Each operator is responsible only for a specific planning task without concern for the consequences of its execution. Operator interactions are handled by the control operators of the architecture. Additional domain operators can be incorporated in the system without having to modify existing operators.
2. *Provide a set of problem-solving operators that may be used in different process planning domains.* Most of the operators described in the EXCAVATION PLANEX prototype were taken directly from the CONSTRUCTION PLANEX system. It is not likely that this will occur in other applications of the PLANEX architecture, but it is probable that operators from different domains will be similar. For example, the *select-technology* operator of HARNESS PLANEX (shown in Figure 5-22) is similar to several domain operators utilized in CONSTRUCTION PLANEX, including the *determine-recommended-duration* operator (Figure 5-18). Most domain operators perform the same tasks (identify a KS to evaluate, evaluate this KS with the KNOWLEDGE SOURCE EVALUATOR, and store the results of the KS evaluation in the context) and thus are similar. Although PLANEX does not provide a unique set of problem-solving operators for different application domains, the development of more generic operators seems plausible.
3. *Incorporate both synthesis and analysis operators.* This requirement has been met as demonstrated by the prototype applications of PLANEX that incorporate both synthesis and analysis operators. Examples of *synthesis* operators are those that generate activities, select technologies and determine activity precedences. Examples of *analysis* operators are those that compute quantities of work, estimate durations and compute earliest and latest event times (e.g., the scheduling algorithms used in CONSTRUCTION PLANEX).
4. *Provide the means to structure operators hierarchically.* Although this capability of the architecture was not utilized in the prototype systems, the architecture provides the means to structure operators into layers as shown in Figure 5-9. Operators in the upper layers (e.g., an operator to compute the cost of the entire building) could trigger the execution of operators in the lower layers (e.g., an operator to compute the cost of one floor) by using an intermediate KS. This strategy might be useful in applications where there are many domain operators.

5.4.3 Control

The following requirements for control of the execution of the problem-solving operators were established. PLANEX should:

1. *Make explicit control decisions that solve the problem.* The control operators of PLANEX are responsible for making explicit decisions about how the problem-solving operators should be executed by analyzing the information stored in the *Domain Operator Schemas* (DOSs). Thus, this requirement is met through the architecture of the system.
2. *Decide what operators to execute in terms of their feasibility and desirability.* This requirement is also satisfied through the capabilities provided for control. PLANEX distinguishes between the feasibility and the desirability of executing a domain operator by using information stored in the DOS. An operator is *feasible* whenever its preconditions are satisfied while it is *desirable* whenever its outcome contributes to solving a goal. The *Backward Search Operator* (BSO) searches for desirable operators until all of the initial and intermediate goals are satisfied and feasible sequence of operators has been identified. In contrast, the *Forward Propagation Operator* (FPO) deals only with feasible domain operators; all feasible operators whose arguments have changed are executed.
3. *Dynamically plan strategic sets of operators.* PLANEX formulates strategic plans of domain operators by applying the control operators as described in Section 4.3. Whenever the user modifies a slot or requests the execution of domain operators whose preconditions are not satisfied, PLANEX modifies the information stored in the agenda. At any point in the planning process, the user may execute the FPO or BSO that will plan the sequence of operators to propagate context changes or achieve goals. Again, the control operators provide the mechanism to satisfy this requirement of the architecture.
4. *Incorporate different control heuristics in the planning process.* Control decisions are based exclusively on the preconditions and effects of the domain operators as described in the DOSs, and all reasoning is based on data availability. Thus, the current architecture does not meet the stated requirements, but the architecture does not preclude other control strategies. PLANEX could incorporate more elaborate control heuristics similar to those of OPM (see p. 32). Alternatively, control operators that use knowledge about the status of the agenda could be added to the architecture.

5.4.4 User Interaction

The following requirements for user interaction were established. PLANEX should:

1. *Provide the means to create, discard or update domain knowledge.* The KNOWLEDGE SOURCE ACQUISITION MODULE of PLANEX provides an interactive environment to create, discard or update domain knowledge, thus meeting the requirement. Displaying KSs in a tabular format facilitates knowledge acquisition and provides transparent process planning knowledge.
2. *Provide the means to modify the set of operators.* PLANEX does not include an explicit mechanism for modifying the set of problem-solving operators of a particular application. The only mechanism available is the editor provided by the host system. Therefore, this requirement is not met in the current version of the system.
3. *Provide the means to control the planning process.* The CONTROL PANEL provides an interactive environment for controlling the planning process. The user may insert goals and changes in the agenda, invoke one or more domain operators and modify operator precedences. In addition, some applications provide menus which can be used to invoke problem-solving operators. With these two tools, the user explicitly controls the planning process.
4. *Explain results in terms of the knowledge used to obtain them.* The relationships between problem-solving operators and KSs provide PLANEX with limited explanation facilities. Each time an operator is executed, a pointer to the KS used by the operator is stored with the application object of the operator. This information is used to provide an explanation of the planning results in terms of the knowledge used. Thus, the architecture has some features which meet the requirement, but a comprehensive explanation facility is lacking.
5. *Provide the means to produce reports with flexible formats.* The REPORT GENERATOR has proven to be flexible enough to produce most of the reports for the application systems. Only a few very complicated reports, such as the *Process Sheet* report of HARNESS PLANEX, require a separate output processor. Thus, the REPORT GENERATOR is a good initial model of a general output tool.
6. *Provide graphic display of results.* PLANEX provides some graphical displays that are applicable in various process planning domains, including the *Activity-On-Node* diagram used in the CONSTRUCTION PLANEX and EXCAVATION PLANEX systems. The architecture provides capabilities to implement and use specific graphical tools in individual applications. However, it is difficult to identify a complete set of generic graphical displays for all PLANEX applications.

6 CONSTRUCTION PLANEX: An Expert System for Construction Project Planning

The construction industry is characterized by separation and isolation of design and project planning [50]. Design of the final facility and construction project planning are typically done by different professionals and different organizations. Design is the responsibility of architects and engineers, whereas project planning is performed by construction contractors who may or may not have engineering training. These different professionals have quite different terminology, different perspectives on the construction process, and—all too often—a mutually antagonistic and disdainful relationship. The traditional bidding process for public projects represents an extreme case of the separation between design and project planning. In this process, architecture and engineering firms develop final facility plans which are made available to numerous general contractors as a set of physical plans and specifications for the completed facility. Each contractor must review the plans to determine the required quantity of work and formulate appropriate construction plans. The formulation of the project plan is done manually, working from the physical plans. The contract for performing the construction is typically awarded to the qualified builder who submits the lowest bid.

Effective computer-based construction project planning systems could be useful in a number of ways. Architecture and engineering design firms could use such systems to improve their designs and estimates of required costs and project durations. Contractors could use these aids to develop better plans more

rapidly and at less cost. Reducing planning costs is particularly important because many plans are developed for bids that are never won. Improved and detailed project plans would be of considerable value in expanding the role of *Computer-Integrated Construction* in areas such as materials procurement and project control. Furthermore, computer-based models of the facility and the construction process can substantially aid communication among the various professionals and organizations involved in the process. Computer-based animations of the construction process are only one of many useful techniques that can be used in this regard.

An example application of PLANEX for planning building construction is CONSTRUCTION PLANEX, a knowledge-based expert system that generates activity plans for the excavation and structural erection of concrete or steel-frame buildings. The intent of system development was to explore the application of knowledge-based systems to the automatic generation of construction project plans. As noted in Section 1.5, the current version of the system is the result of a multi-cycle refinement process which also resulted in the development of the PLANEX system architecture. Knowledge for the system was obtained from both the literature and an experienced construction planner [4].

The previous chapter presented an overview of how the basic components of the PLANEX architecture are used in CONSTRUCTION PLANEX. This chapter describes the structure and behavior of CONSTRUCTION PLANEX in detail and illustrates the use of the system with examples. The models used by the CONSTRUCTION PLANEX system in each of the stages of the construction planning process are presented, followed by a discussion of the components of the system and the relationships among them. Readers interested in only a conceptual understanding of the system can skip Section 6.2. Section 6.3 illustrates the use of the system both as a stand-alone planning assistant and as a component of an integrated computing environment for building design. The chapter concludes with a complete example.

6.1 Models for Construction Planning Used in the System

Several models for different elements of the construction planning process were discussed in Chapter 2. Models were classified into the following categories:

- *Definition of Work Tasks and Precedence Relationships*—models for the generation of the project activity network;
- *Choice of Technologies and Construction Methods*—models for the selection of technologies and methods to perform construction activities;
- *Estimation of Activity Durations and Costs*—models to estimate activity durations and costs based on the selected technologies and construction methods; and

- *Preparation and Maintenance of Project Schedules*—models to compute project schedules that satisfy the time and resource constraints of the project.

In generating a construction plan for a building, CONSTRUCTION PLANEX uses models from each of these four categories. This section describes these models and discusses the applicability and limitations of each.

6.1.1 Definition of Work Tasks and Precedence Relationships

In generating a project activity network, CONSTRUCTION PLANEX uses the “bottom-up” activity formulation model presented in Section 3.2.1. The activity network is built in four steps:

- Step 1. Model Building.* The building is described in terms of unitary components called *design elements*. Example components are beams, slabs, columns, walls, footings and diagonals (bracing).
- Step 2. Identify Activities.* The system determines the *element activities* required to construct each design element. Example element activities are form placement for individual concrete components and excavation of individual column footings.
- Step 3. Aggregate Activities.* The system aggregates element activities into *project activities*. Example project activities are the excavation of all footings elements and concrete placement on a particular floor.
- Step 4. Link Activities.* The system creates a project network by establishing precedence links among project activities. For example, the system creates a link from a form placement activity to the corresponding concrete pouring activity for an element.

CONSTRUCTION PLANEX has some limitations regarding how the bottom-up activity formulation model is used:

- CONSTRUCTION PLANEX generates plans using detailed design information describing the structural elements of a building (e.g., dimensions, type of materials). The system cannot generate plans from more abstract building design descriptions.
- The system only generates activities associated with the construction of the design elements. To generate other activities, artificial design elements must be created (e.g., an object describing the geometric characteristics of the site is needed to generate a site-clearing activity).
- CONSTRUCTION PLANEX first identifies the activities and then establishes precedences. Alternative knowledge sources or operators could perform these tasks simultaneously.
- Activities are divided into two levels of aggregation: (1) *element activities* representing activities used to construct individual design elements; and

(2) *project activities* representing groups of element activities on a particular floor of the building. Sophisticated spatial aggregation of the construction activities would be useful.

6.1.2 *Choice of Technologies and Construction Methods*

CONSTRUCTION PLANEX assigns *crews* to activities after formulating the project activity network and before estimating activity durations and costs. Each crew is a combination of both labor and equipment and is represented using a *crew schema*. Crew schemas describe the crew components, standard crew productivities and average unit costs. Figure 6-1 shows the schema used to represent a crew named "crew-excavation-foundation-05". This crew is composed of a backhoe with a 3/4 cubic yard bucket and a single machine operator. The standard productivity of this crew is 100 cubic yards per day, and its average unit costs are \$6.74 per cubic yard of normal time work and \$10.11 per cubic yard of overtime work.

CONSTRUCTION PLANEX models the technology selection process in two steps:

- Step 1. *Select Crew Type.* The system chooses a crew for each project activity by using information about site characteristics, design elements and the activity. In this step, CONSTRUCTION PLANEX assumes that each activity can be performed with the most appropriate crew.
- Step 2. *Determine the Number of Crews to be Used.* Once the crew type has been selected, the system computes a recommended activity duration as a function of the quantity of work. Then the system decides how many crews are needed to perform the activity on the basis of the recommended activity durations. (Selected crew allocations may be modified by the user.) In this step, no time-cost trade-offs are analyzed.

```
(defschema crew-excavation-foundation-05
;
; Crew for excavating sand and gravel soil with
; 3/4 cubic-yard backhoe [71, p. 37].
;
  (is-a      crew)
  (component-names ((1 backhoe-3/4)
                     (1 operator-backhoe-3/4)))
  (std-productivity 100.0)
  (prod-unit      cu-yd/day)
  (normal-cost    6.74)
  (overtime-cost  10.11))
```

Figure 6-1. Example of a Crew Schema

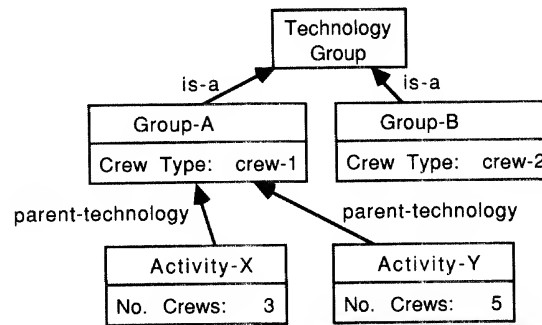


Figure 6-2. Representational Structures for Storing Technology Decisions

The results of the technology selection process are stored as shown in Figure 6-2. The values of the number of crews are stored in activity schemas and the crew types are stored in *technology-group* frames. Each technology group object is linked to all of the activities using the same crew type. This structure lets the system identify all activities impacted by a technology change. Assume that the system allocates the crew shown in Figure 6-1 to various excavation activities and the user later decides to change this assignment. The information stored in the corresponding technology group object permits CONSTRUCTION PLANEX to identify the activities affected by this decision. The system asks the user if the change is local to an activity or if it should be propagated to all the activities which use the same crew (i.e., linked to the technology group object), and makes either the corresponding local change or propagates the effect.

Dividing the technology selection process into two steps assumes that the selection of the crew type can be made without considering the number of crews allocated to the activity. CONSTRUCTION PLANEX does not combine technology selection with activity scheduling. This simplified model is appropriate for obtaining an initial project plan and reflects contractors' common practice of separating the decisions of crew type and number of crews [4, p. 4]. If the system were to combine these two steps, a model similar to the Decision CPM (see p. 44) would be needed. In this case, the two-step strategy would be used to generate a Decision CPM network incorporating appropriate combinations of crew types and number of crews.

6.1.3 Estimation of Activity Durations and Costs

CONSTRUCTION PLANEX estimates durations and costs of project activities using simple models based on average productivities and unit costs [50]. The process for computing activity durations is shown in Figure 6-3. First, the system

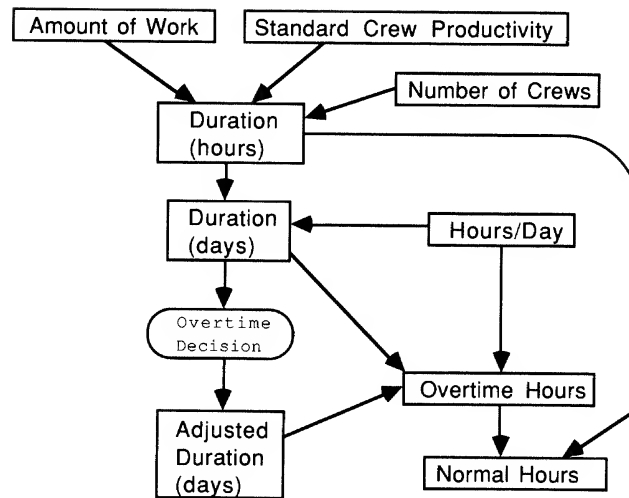


Figure 6-3. Process for Activity Duration Estimation

computes the expected activity duration (in hours) by dividing the total quantity of work by the number of crews allocated to the activity and the standard productivity of each crew. Then the system divides the expected duration by the number of normal working hours per day to obtain the estimated activity duration in days. Following this computation, CONSTRUCTION PLANEX eliminates fractional days if overtime is permitted. (If overtime is not used, the duration is increased one full day.) Finally, the system subtracts the number of overtime hours from the estimated activity duration in hours to obtain the number of normal working hours.

The activity labor cost is computed by adding its normal and overtime costs. The normal working cost is calculated by multiplying the unit crew cost per normal working hour by the number of normal working hours and the number of crews. Similarly, the overtime cost is computed as the product of the unit crew cost per overtime hour and the number of overtime hours and the number of crews.

A desirable extension of the CONSTRUCTION PLANEX system would be to replace the simple estimating procedure described above with a more elaborate hierarchical estimator that uses the concepts from MASON (see p. 59). Also, it may be desirable to compute not only the expected values of the activity duration and costs, but also some measure of their variability. This would allow the system to use probabilistic scheduling models such as the PERT for estimating expected project durations.

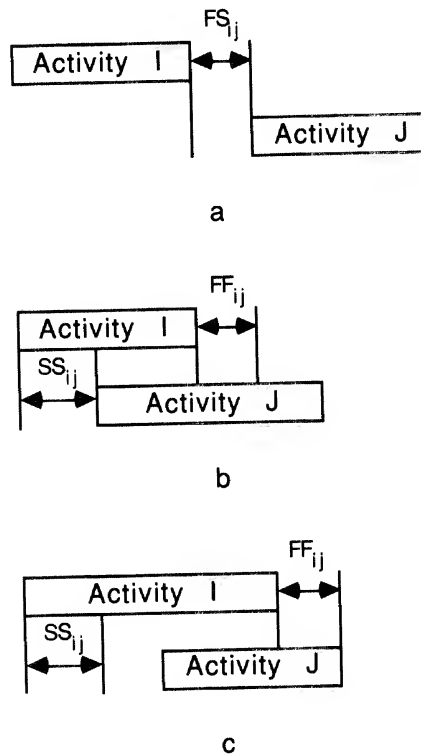


Figure 6-4. Example Precedences Between Consecutive Project Activities

6.1.4 Preparation and Maintenance of Project Schedules

In generating project schedules, CONSTRUCTION PLANEX uses the *unified activity network* model described in Section 3.2.2. This model can be used to compute the earliest and latest event times in networks with multiple types of precedences and windows constraints.

Figure 6-4 shows three common types of precedence relationships between consecutive project activities. In case a, the succeeding activity *J* cannot start until the preceding activity *I* has finished completely, a *Finish-to-Start* or *FS* relationship. Such a link can be used to represent that pouring concrete in a particular floor will start after all the reinforcement for that floor has been placed. In cases b and c, the following activity can start after a portion of the preceding activity has been completed. Case b, called *Start-to-Start* or *SS*, is common in *fast-track* schedules where succeeding activities overlap. For example, placing wallboard can start after only some of the wall studs are in place.

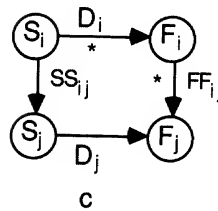
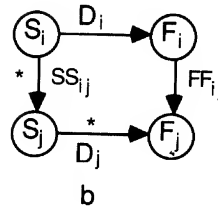
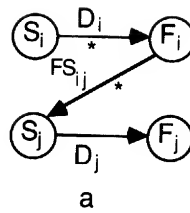


Figure 6-5. Representation of Precedences in the Unified Activity Network Model

When activities overlap, additional *Finish-to-Finish* or FF links may be used to ensure that following activities are performed continuously. For example, in case c, the FF link delays the start of activity J more than the SS value in order to finish this activity after activity I has been completed.

Figure 6-5 shows the unified activity network model representation of the three precedence cases of Figure 6-4. One node is used to represent the start event of the activity and another node is used to represent the finish event. Links represent activity durations, precedence relationships and window constraints. For case a, the links representing the duration of activity I and the FS precedence are critical (marked with an asterisk [*]) because increasing either duration will increase the earliest completion time of the project. In case b, the SS link and the duration of activity J are critical (e.g., increasing the duration of activity J will increase the duration of the project). In case c, the duration of activity I and the FF link are critical, which means that increasing the duration of activity J will not affect the completion time of the project, but will require the activity to start sooner (as long as it is not in conflict with the SS precedence).

6.2 System Architecture

As described in Chapter 5, the CONSTRUCTION PLANEX system is implemented using the four components of the PLANEX architecture:

1. *representational structures* store information about the site, the building to be constructed, the activities involved in the project and the resources available to perform these activities;
2. *problem-solving operators* perform construction planning tasks such as technology choice, activity synthesis, duration estimation, etc.;
3. *knowledge sources* provide construction knowledge for the operators; and
4. *user interface mechanisms* provide the means to control the execution of the construction planning process and allow the user to modify or obtain information about planning decisions.

Each of these components of the system is described below.

6.2.1 Representational Structures

CONSTRUCTION PLANEX uses the three basic representational structures shown in Figure 6-6. The figure is only illustrative since additional levels or aggregations are used as needed. The basic structures are:

- *Tree of Design Elements.* The building is described in terms of individual design element schemas that represent building components such as beams or columns. These schemas are aggregated by material (concrete or steel), element type (e.g., columns) and location (e.g., first floor).
- *Tree of Element Activities.* The activities used to construct each of the design elements are aggregated using an extension of the MASTERFORMAT [18] coding system⁹ into a tree of element activities. The different levels of the MASTERFORMAT (Division, Broadscope and Narrowscope) plus the extensions are described below.
- *Tree of Project Activities.* Construction activities are aggregated by the type of activity represented (e.g., formwork) and the type of design element associated with the activity (e.g., formwork for columns versus formwork for beams).

Building components and construction activities are also aggregated with respect to their location in the site. There are four levels of spatial aggregation:

⁹ The MASTERFORMAT coding system was developed by the *Construction Specifications Institute* and is widely used by architects, engineers, contractors and suppliers for categorizing information related to the construction process.

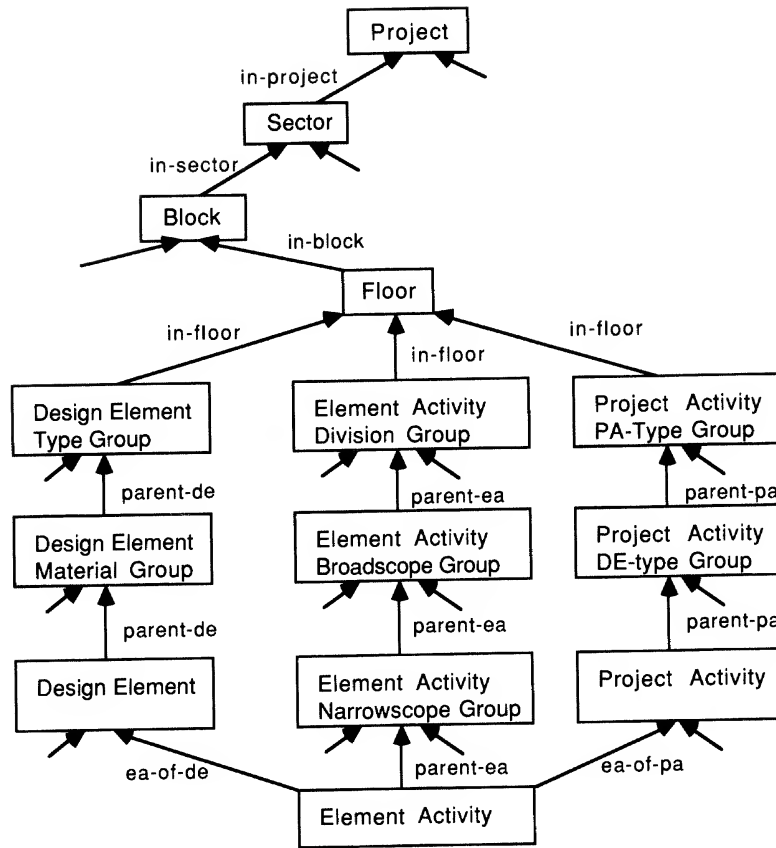


Figure 6-6. Representational Structures of CONSTRUCTION PLANEX

- *floor* groups all objects of a particular story of a building;
- *block* groups all objects of a single building;
- *sector* groups all objects of a site sector; and
- *project* groups all objects of a particular project.

Representational structures are identified by hierarchical coding systems consistent with the levels shown in Figure 6-6. Separate hierarchical codes exist for design elements and activities. For example, an element activity might be named *F00-EA-2-220-10-1* which has the following interpretation:

- “F00” is the root node of element activities on a particular floor (in this case, the foundation is denoted as floor 00);
- “EA” indicates that the schema is an element activity;

- “2” indicates that the activity is sitework;
- “220” identifies the activity is excavation;
- “10” indicates that the activity is for column footings; and
- “1” identifies the group of column footings (in this case group “1”).

The codes “2-220-10” correspond to the division group, broadscope group and narrowscope group of the MASTERFORMAT coding system.

Design elements are identified by floor, element type, material type and group number. For example, the design element *F00-DE-60-1-1* is interpreted as:

- “F00” is the root node of the hierarchy specifying the foundation level of the structure;
- “DE” indicates the schema is a design element;
- “60” indicates the element type is column footings;
- “1” indicates the material type is concrete; and
- “1” indicates element is the first of the group.

Project activities are coded similarly to design elements except they are identified as *PA* schemas rather than element activities (*EA*) or design elements (*DE*). The code consists of the location, activity type, design element type and group number. The codes for the element type, activity type and material type are arbitrary, but based on similar codes used by a contractor [4]. In general, the activity code corresponds to the MASTERFORMAT narrowscope group code. Example project activity codes (activity and design element type) are shown in Figure 6-7.

Codes may be more or less detailed than the examples described. For example, if a set of related design elements are independent of material (e.g., earthwork), the material level may be omitted from the code. Similarly, in some instances the lowest level group code is omitted if the item refers to all groups. Alternatively, in many instances the element activity code includes the design element type and material type after the MASTERFORMAT group codes and before the group number. The structure of the various operators and knowledge sources are such that CONSTRUCTION PLANEX can readily identify and deal with all such cases. The remainder of this section describes a simplified set of codes consistent with Figure 6-6.

All codes begin with the same set of one or more spatial aggregation codes. In these examples, the code begins with a location (i.e., floor) designator (e.g., “F00”) which acts as a *root* node for the hierarchies of design elements and activities. Each floor has a different root node identifier (“F01”, “F02”, etc.). These floor designators are also associated with particular blocks, sectors and projects (e.g., *P01-B00-S00-F00-...*). Thus, even though the example codes themselves do not reflect these links, aggregation to larger entities is possible. In some instances, CONSTRUCTION PLANEX will generate codes with the full spatial location prefix.

CODE	ACTIVITY NAME
10-60	EXCAVATION-FOUNDATION
15-60	HAUL-EXCAVATION-FOUNDATION
17-60	BACKFILL-FOUNDATION
20-60	FORMWORK-FOUNDATION
30-60	REMOVE-FORMS-FOUNDATION
40-60	REINFORCING-STEEL-FOUNDATION
50-60	POUR-CONCRETE-FOUNDATION
20-65	FORMWORK-COLUMNS
30-65	REMOVE-FORMS-COLUMNS
40-65	REINFORCING-STEEL-COLUMNS
50-65	POUR-CONCRETE-COLUMNS
55-65	ERECTION-STEEL-COLUMNS-DIAGONALS
20-80	FORMWORK-SLABS
30-80	REMOVE-FORMS-SLABS
40-80	REINFORCING-STEEL-SLABS
50-80	POUR-CONCRETE-SLABS
55-81	ERECTION-STEEL-BEAMS
58-81	JOIN-STEEL

Figure 6-7. Sample Project Activities and Codes

Similar codes are used to identify knowledge sources and operators which correspond to particular design elements or activities. For example, the knowledge source used to generate element activities required to construct concrete column footings (such as *F00-DE-60-1-1*) is *KS-Create-EA-60-1*, which is interpreted as:

- “KS” identifies the schema as a knowledge source;
- “Create-EA” identifies the knowledge source as a schema containing rules to create element activities used by the *Create-EAS* operator; and
- “60-1” identifies the design element type and material (concrete column footings) to which this knowledge source is applied.

6.2.1.1 Tree of Design Elements Design element schemas are used to describe structural building components. CONSTRUCTION PLANEX can plan the excavation and erection of the following types of design elements:

- concrete column footings;
- concrete or steel beams;
- concrete or steel columns;
- concrete or steel diagonals (bracing); and
- concrete floor slabs.

At the start of the planning process, CONSTRUCTION PLANEX is given a file containing design element schemas describing the building (with some slots empty). The system organizes these schemas into a tree structure using the

```

(defschema F00-DE-60-1-1
;-----Relationship slots
  (is-a      de)
  (parent-de F00-DE-60-1)
  (de-has-eas
    F00-EA-2-220-40-60-1-1
    F00-EA-3-110-20-60-1-1
    F00-EA-3-310-10-60-1-1
    F00-EA-3-210-00-60-1-1
    F00-EA-3-110-10-60-1-1
    F00-EA-2-225-10-60-1-1
    F00-EA-2-220-10-60-1-1)
;-----Classification slots
  (type-de      60)
  (type-material 1)
  (de-code      60-1)
  (name-de      column-footings-1)
  (number-de    1)
;-----Multiplier slot
  (multiplier 4)
;-----Location slots
  (project      p01)
  (sector       s00)
  (block        b00)
  (floor        f00)
  (root-code    F00)
  (xg-coordinate (10 15 20 25))
  (yg-coordinate (10 10 10 10))
  (zg-coordinate (-4 -4 -4 -4))
;-----Geometry slots
  (xl-dimension 10)
  (yl-dimension 8)
  (zl-dimension -1.50)
;-----Specifications slots
  (const-type    cast-in-place)
  (concrete-type normal-weight-3000)
  (psteel        4.0)
;-----Explanation slot
  (why-eas       KS-Create-EAS-60-1))

```

Figure 6-8. Example of a Design Element Schema

Create-DE-Tree operator described in Section 6.2.2.1. Elements are grouped in a bottom-up manner. For example, the *F00-DE-60-1-1* schema of Figure 6-8 is linked below the schema *F00-DE-60-1* which groups all concrete column footings of various dimensions (i.e., aggregated by material type). This schema is linked to a parent schema *F00-DE-60* which groups all column footings of the building (i.e., aggregated by design element type). Thus, the design elements are grouped into four levels as shown in Figure 6-6: level 1 is all design elements at a given location; level 2 is all design elements of a given type at that

location; level 3 is all design elements of a specific material type for the design element type; and level 4 groups individual elements of a specific material, type and location.

An example of a design element schema is shown in Figure 6-8. This schema is titled *F00-DE-60-1-1* and stores information detailing a group of four column footings of identical dimensions and materials. The schema contains the following types of slots:

- *Relationship* slots represent links among the design element schema and other schemas of the context. The figure shows three relationship slots: (1) the *is-a* slot identifies the schema as a design element schema; (2) the *parent-de* slot indicates the parent schema in the tree of design elements; and (3) the *de-has-eas* slot stores the names of the element activities required to construct the column footings.
- *Classification* slots identify the type of design element. Four slots are used for this purpose: (1) the *type-de* slot defines the type of design element ("60" indicates column footings); (2) the *type-material* slot defines the type of material used to construct the design element ("1" indicates concrete); (3) the *name-de* slot provides a name for the design element ("column-footings-1"); and (4) the *number-de* slot indicates the group number ("1" indicates that this is the first group of column footings). In addition, the *de-code* slot is the code of the design element and contains the concatenation of the values of the *type-de* and *type-material* slots.
- *Multiplier* slot stores the number of identical column footings represented in the schema. The example schema represents a group of four identical column footings.
- *Location* slots specify the *project*, *sector*, *block* and *floor* location of the design elements. The *xg-coordinate*, *yg-coordinate* and *zg-coordinate* slots specify the global x, y and z coordinates of a datum point on each column footing. The *root-code* slot stores the spatial aggregation code used to identify schemas.
- *Geometry* slots describe the geometric characteristics of the design element. For example, the *xl-dimension*, *yl-dimension* and *zl-dimension* slots store the x, y and z dimensions of the footing.
- *Specifications* slots contain descriptive information about the design element that is relevant to the planning process. For example, the *const-type* and the *concrete-type* slots specify construction technologies. The *psteel* slot specifies the percentage of reinforcing steel for the design element.
- *Explanation* slot stores the names of the knowledge sources used to compute values in the schema. In the example, the *why-eas* slot indicates that the KS titled *KS-Create-EAS-60-1* was used to determine the names of the element activities required to build a column footing.

6.2.1.2 Tree of Element Activities Element activity schemas are used to describe activities performed in the construction of the design elements of the building. These schemas are organized using a coding system that is an extension of the standard MASTERFORMAT coding system. As noted above, the standard MASTERFORMAT codes have been extended to incorporate information specifying the type of design element for the activities. Element activity codes of CONSTRUCTION PLANEX have four basic parts:

1. *division number* identifies the general type of activity and corresponds to the *division* level of the MASTERFORMAT;
2. *broadscope number* identifies the subtype of activity under each division;
3. *narrowscope number* provides a third level of classification with respect to the materials or building elements associated with the activity; and
4. *design element number* identifies a specific design element or group of design elements to which the element activity is applied. This value consists of up to three parts: (1) the design element type; (2) the material type; and (3) the element group number. As described above, the group number is dropped if the code pertains to all elements, or the design element type and material type codes may be dropped if the activity is independent of the design element.

These element activity codes are used to group element activities into a tree with the same levels of hierarchy as those shown in Figure 6-6. For example, groups of element activities representing the excavation of column footings are aggregated below a narrowscope group "2-220-10" ("Excavation, Backfilling and Compacting of Structures"), which is linked to a broadscope group "2-220" ("Excavation, Backfilling and Compacting"), which is grouped below a division group "2" ("Sitework"). Together these elements form the five levels shown in Figure 6-6: level 1 is all element activities at a given location; level 2 is all element activities of a given division at that location; level 3 is the set of all element activities with a specific broadscope group number within the division; level 4 is all activities of a narrowscope code within the broadscope; and level 5 is the individual element activities of a specific narrowscope, broadscope, division and location. The aggregation information is stored in the *parent-ea* slot of the element activity schemas. The tree is built by the *Create-EA-Tree* operator described in Section 6.2.2.2.

Figure 6-9 shows an example of an element activity schema. This schema is titled *F00-EA-2-220-10-1* and stores the description of the excavation activity required for one of the column footings of group "column-footings-1" (i.e., design element *F00-DE-60-1-1*). The schema contains the following types of slots:

- *Relationship* slots represent links between the element activity schema and other objects of the context. Four relationship slots are used: (1) the *is-a* slot

```

(defschema F00-EA-2-220-10-1
;-----Relationship slots
  (is-a ea)
  (parent-EA F00-EA-2-220-10)
  (ea-of-DE F00-DE-60-1-1)
  (ea-of-PA F00-PA-10-60)
;-----Classification slots
  (ea-code 2-220-10-1)
  (ea-name excavate-column-footings-1)
;-----Quantity Take-Offs slots
  (amount-of-work-ea 24.0)
  (unit-of-measure cu-yd)
;-----Material slot
  (material-package none)
;-----Duration slot
  (duration-ea 0.1)
;-----Explanation slots
  (why-amount-formula formula-02)
  (why-amount-ks KS-Amount-2-220-10)
  (why-project KS-PA-2-220-10))

```

Figure 6-9. Example of an Element Activity Schema

identifies the schema as an element activity schema; (2) the *parent-ea* slot stores the name a parent schema in the tree of element activities; (3) the *ea-of-de* slot stores the name of the design element schema associated with the activity; and (4) the *ea-of-pa* slot stores the name of the aggregated project activity which includes this element activity.

- *Classification* slots identify the type of element activity. Two slots are used for this purpose: (1) the *ea-code* slot stores the activity code of the element activity; and (2) the *ea-name* slot stores the name of the element activity.
- *Quantity Take-Offs* slots store the amount of work needed to construct the element activity and the unit in which this amount is measured.
- *Material* slot indicates the name of material package used to perform this activity. The material package is the set of all temporary and permanent construction materials used in building the associated design element. In the example, no materials are required to perform the activity because it is an excavation activity.
- *Duration* slot stores the estimated duration of the element activity.
- *Explanation* slots store the names of the knowledge sources or formulas used to fill the slots of the schema. The example shows three explanation slots: (1) the *why-amount-KS* slot indicates that the KS named *KS-Amount-2-220-10* was used to select the formula which computed the amount of the activity; (2) the *why-amount-formula* stores the name of this formula; and (3) the *why-project* slot stores the name of the KS used to create the project activities.

6.2.1.3 Tree of Project Activities Project activity schemas represent aggregations of element activities. CONSTRUCTION PLANEX creates project activity schemas to obtain a reasonable level of granularity in the activity network. These schemas are grouped with respect to their project element type, design element type and location into a tree of project activities similar to the trees of design elements and element activities described above.

The tree structure is generated by the *Create-PA-Tree* operator described in Section 6.2.2.3. Project elements are grouped into four levels (shown in Figure 6-6): level 1 is all project activities at a given location; level 2 is all project activities of a given project activity type at that location; level 3 is all project elements of a specific design element type for the specific project activity type; and level 4 is an individual project activity of a specific design element type, project activity type and location.

An example of a project activity schema is shown in Figure 6-10. The schema represents the activity to remove forms from the first floor of a building. It has the following types of slots:

- *Relationship* slots are used to link the project activity schema to other objects of the context. Four relationship slots are used: (1) the *is-a* slot identifies the schema as a project activity schema; (2) the *parent-pa* slot stores the name of a parent schema in the tree of project activities; (3) the *pa-has-eas* slot stores the names of the element activity schemas which comprise this project activity; and (4) the *parent-technology* slot identifies the name of the technology group object which stores the crew type for this project activity.
- *Classification* slots identify the type of project activity. Two classification slots are used: (1) the *pa-code* slot stores the activity code of the project activity; and (2) the *pa-name* slot stores the name of the project activity.
- *Quantity Take-Offs* slot stores the work quantity used to perform the project activity. This value is the sum of the amounts of work (i.e., *amount-of-work-ea* slot) of the element activities aggregated in this project activity.
- *Duration* slots store information describing the duration of the project activity. Data stored includes the recommended and computed durations, upper and lower duration bounds, and the distribution of normal and overtime hours.
- *Technology* slots store the number of crews used to perform the activity and the adjusted productivity of these crews.
- *Precedence* slots store the names of the successors of an activity, the types of links to each successor and the leads or lags for each of these links. The *successors* slot contains all unique activity successors while the *succs* slot contains one successor for each type of precedence link. The project activity shown in the figure has no successors.
- *Cost* slots store the crew and material costs associated with the project activity. These costs are in current dollars and do not include overhead or

```

(defschema F01-PA-30-65
✓ ;-----Relationship slots
    (is-a                                pa)
    (parent-pa                          F01-PA-30)
    (pa-has-eas                         F01-EA-3-110-20-65-1-1
                                         F01-EA-3-110-20-65-1-2
                                         F01-EA-3-110-20-65-1-3
                                         F01-EA-3-110-20-65-1-4)
    (parent-technology                 group-technology-4)
✓ ;-----Classification slots
    (pa-code                           30-65)
    (pa-name                           remove-forms-columns-F01)
✓ ;-----Quantity Take-Offs slot
    (amount-of-work-pa                 5040.0)
✓ ;-----Duration slots
    (recommended-duration              4)
    (duration                          3.0)
    (low-dur                           3.0)
    (high-dur                          3.15)
    (normal-hours                      24.0)
    (overtime-hours                    1.2)
;-----Technology slots
    (number-crews                      4.0)
    (adj-productivity                  400.0)
;-----Precedence slots
    (successors                        none)
    (succs                             none)
    (link                              fs)
    (low-lags                          0)
    (high-lags                         *p-inf*)
;-----Cost slots
    (total-cost-crew                   619.2)
    (cost-crew-per-day                 206.4)
    (total-cost-materials              0.0)
    (cost-materials-per-day            0.0)
    (overall-total-cost                619.2)
    (overall-cost-per-day              206.4)
;-----Scheduling slots
    (est                              115.22)
    (eft                              118.22)
    (lst                              210.27)
    (lft                              213.27)
;-----Explanation slots
    (why-duration                      KS-Dura-30-65)
    (why-successors                    KS-Succ-30-65)

```

Figure 6-10. Example of a Project Activity Schema

profit. This category has six slots: (1) the *total-crew-cost* slot stores the total labor and equipment cost for the activity; (2) the *cost-crew-per-day* slot stores the daily crew cost; (3) the *total-cost-material* slot stores the total material cost; (4) the *cost-materials-per-day* slot stores the daily material cost; (5) the *overall-total-cost* slot is the sum of the total material and crew costs; and (6) the *overall-cost-per-day* is the sum of the daily material and crew costs.

- *Scheduling* slots describe the schedule for the project activity. There are four schedule slots: (1) the *est* slot is the earliest-start-time of the activity; (2) the *eft* slot is the earliest-finish-time of the activity; (3) the *lst* slot is the latest-start-time of the activity; and (4) the *lft* slot is the latest-finish-time of the activity.
- *Explanation* slots store the names of the knowledge sources used in computing values in the schema. The example shows two explanation slots: (1) the *why-duration* slot indicates that the KS called *KS-Dura-30-65* was used to determine the recommended duration; and (2) the *why-successor* slot stores the name of the KS used to determine the successors.

6.2.2 Problem-Solving Operators

CONSTRUCTION PLANEX includes three basic types of problem-solving operators:

- *design element* operators;
- *element activity* operators; and
- *project activity* operators.

This classification is based on the type of context objects that the operators take as arguments. For example, the *Get-Duration-PAS* operator is applied to project activity schemas. Therefore, it is a project activity operator.

The following presentation describes each class of operators in detail, including a discussion of the procedures used by the operators, the *Domain Operator Schemas* (DOSs) which describe the preconditions and effects of the operators and the designation of which types of Knowledge Sources (KSs) are evaluated when the operator is executed.

6.2.2.1 Design Element Operators CONSTRUCTION PLANEX has two operators that are applied to design elements:

- *Create-DE-Tree* aggregates design element schemas in the bottom-up manner described in Section 6.2.1 and produces the tree of design element schemas; and
- *Create-EAS* creates element activity schemas describing the tasks used to construct a design element.

```

(defschema Create-DE-Tree
  (is-a          operator)
  (domain-type   de)
  (application-object current-object)
  (input-objects current-object current-object
                 current-object)
  (input-slots   is-a type-de type-material)
  (input-bindings nil nil nil)
  (input-cond-types filled filled filled)
  (output-objects current-object current-object)
  (output-slots   parent-de de-code)
  (output-bindings nil nil)
  (output-predictable yes yes)
  (output-effect-type fill fill))

```

Figure 6-11. Domain Operator Schema for the *Create-DE-Tree* Operator

Create-DE-Tree. The *Create-DE-Tree* operator creates the tree of design elements described in Section 6.2.1.1. When applied to a design element schema, this operator performs the following steps:

- Step 1.* Create the name of the immediate parent schema in the design element tree by trimming the last identifier from the name of the daughter schema. For example, design element *F00-DE-80* is the parent of schema *F00-DE-80-1*.
- Step 2.* If the parent schema does not exist: create it.
- Step 3.* Link the daughter design element schema to the parent schema.

The operator repeats these three steps until the root of the tree is reached. This root corresponds to the project schema (e.g., *F00*).

Figure 6-11 shows the *Domain Operator Schema* (DOS) of the operator. When linking a new design element into the tree of design elements, this operator requires that the *is-a* slot of the new element be filled in order to check that the schema is a design element (the value of the *is-a* slot is "de"). The effect of the operator is to fill the *parent-de* and *de-code* slots of the design element schema being linked into the tree. The *de-code* is created by concatenating the values of the *type-de* and the *type-material* slots and is used by other domain operators. Thus, the operator requires that the *type-de* and *type-material* slots are filled.

Create-EAS. The *Create-EAS* operator is responsible for synthesizing the element activities used to construct a particular design element. Element activities associated with a design element are generated by evaluating a KS associated with the specific design element. When applied to a particular design element, this operator performs the following steps:

```

(defschema Create-EAS
  (is-a          operator)
  (domain-type   de)
  (application-object current-object)
  (input-objects current-object current-object)
  (input-slots    parent-de de-code)
  (input-bindings nil nil)
  (input-cond-types filled filled)
  (output-objects current-object <eas> <eas>)
  (output-slots    de-has-eas is-a ea-of-de)
  (output-bindings <eas> nil nil)
  (output-predictable yes no no)
  (output-effect-type fill fill fill))

```

Figure 6-12. Domain Operator Schema for the *Create-EAS* Operator

- Step 1.* Build the name of the *Element Activity* KS (see p. 211) to be evaluated by concatenating the prefix *KS-Create-EA-* with the design element code of the element activity (type and material, e.g., the value of the *de-code* slot).
- Step 2.* Evaluate this KS and store the results in a temporary list. Each member of this list is a pair (*ea-schema-name ea-name*) which represents one activity used to build the design element. The pair consists of the name of the schema describing the element activity and the name of the activity itself.
- Step 3.* For each element in the list of results, create the corresponding element activity schema, store the name of the activity in this schema and link the activity schema to the design element object using the *ea-of-de* relationship.

The DOS that describes the *Create-EAS* operator is shown in Figure 6-12. The operator requires that the design element be linked in the tree of design element schemas (the *parent-de* slot must be filled) and have a value in its *de-code* slot. This *de-code* value is used by the operator to identify the name of the element activity KS to be evaluated. Some of the effects of this operator are unpredictable because the names of the element activity schemas are not known until the particular KS is evaluated and generates the list of activities and element activity schema names. Thus, the control operators cannot determine all of the context changes that result from executing the operator.

The DOS schema of Figure 6-12 includes only those effects and preconditions that are relevant for control purposes (i.e., the values in the corresponding slots are used by other domain operators). Other effects, such as filling the name of the element activity, are not represented.

```
(defschema Create-EA-Tree
  (is-a          operator)
  (domain-type   ea)
  (application-object current-object)
  (input-objects current-object)
  (input-slots    is-a)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object current-object)
  (output-slots    parent-ea ea-code)
  (output-bindings nil nil)
  (output-predictable yes yes)
  (output-effect-type fill fill))
```

Figure 6-13. Domain Operator Schema for the *Create-EA-Tree* Operator

6.2.2.2 Element Activity Operators CONSTRUCTION PLANEX includes six operators applied to element activity schemas:

- *Create-EA-Tree* creates the tree of element activities;
- *Compute-Amount-EAS* computes the quantity take-offs;
- *Determine-Unit-EAS* identifies the units of measure for quantity take-offs;
- *Determine-Material-EAS* selects material packages for element activities;
- *Create-PAS-for-EAS* groups element activities into project activities; and
- *Get-Duration-EAS* estimates the duration of element activities.

Create-EA-Tree. The *Create-EA-Tree* operator links an element activity into the tree of element activities described in Section 6.2.1.2. It is similar to the *Create-DE-Tree* operator. Starting with an element activity at the bottom of the tree, it creates parent schemas successively until the root of the tree is reached. Thus, an element activity schema titled *F00-EA-3-110-10-60-1* is linked to its immediate parent, *F00-EA-3-110-10-60*; this schema is then linked to schema *F00-EA-3-110-10*, which is linked to *F00-EA-3-110*, which is linked to *F00-EA-3*, which is linked to the root of all element activity schemas *F00-EA* (for a particular floor location, e.g., *F00*). Three steps are performed repeatedly until the root of the tree is reached:

- Step 1.* Create the name of the immediate parent schema in the element activity tree by trimming the last identifier from the name of the daughter schema.
- Step 2.* If the parent schema does not exist: create it.
- Step 3.* Link the daughter element activity schema to the parent schema.

Figure 6-13 shows the DOS for the *Create-EA-Tree* operator. This schema resembles the DOS of the *Create-DE-Tree* operator. However, the only precondition is that the *is-a* slot of the element activity is filled to verify that the

```

(defschema Compute-Amount-EAS
  (is-a          operator)
  (domain-type   ea)
  (application-object current-object)
  (input-objects current-object current-object <de>
                 <de> <de>)
  (input-slots   ea-code ea-of-de xl-dimension
                 yl-dimension zl-dimension)
  (input-bindings nil <de> nil nil nil)
  (input-cond-types filled filled filled filled filled)
  (output-objects current-object)
  (output-slots   amount-of-work-ea)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))

```

Figure 6-14. Domain Operator Schema for the *Compute-Amount-EAS* Operator

schema is an element activity (the *is-a* slot has a value equal to "ea"). The operator fills the slots *parent-ea* and *ea-code* of the element activity. The *ea-code* slot value is taken from the element activity schema name.

Compute-Amount-EAS. The *Compute-Amount-EAS* operator computes the quantity take-off for an element activity in four steps:

- Step 1.* Build the name of the *Amount* KS (see p. 213) to be evaluated by concatenating the prefix *KS-Amount-* with the element activity code (division, broadscope and narrowscope codes, i.e., the *ea-code* slot value).
- Step 2.* Evaluate this KS and return the name of the formula to be used in the computation.
- Step 3.* Evaluate the quantity take-off formula, inheriting the dimensions of the design element from the design element schema which corresponds to the element activity.
- Step 4.* Store the amount of work in the *amount-of-work-ea* slot of the element activity schema and the name of the formula and KS used in the *why-formula* and *why-amount* slots.

The DOS of the *Compute-Amount-EAS* operator is shown in Figure 6-14. This operator uses information from: (1) the *ea-code* and *ea-of-de* slots of the element activity schema; and (2) the *geometry* slots of the design element schema associated with the element activity. In the DOS, only the *xl-dimension*, *yl-dimension* and *zl-dimension* slots are included. However, some quantity take-off formulas require other data such as the percentage of reinforcing steel or the type of concrete. For other activities, only some of the geometry slots are needed (e.g., for steel beams, only the length and the section designation are

relevant). A solution to this problem is multiple DOSs, one for each case (e.g., *Compute-Amount-EAS-Steel* and *Compute-Amount-EAS-Concrete*).

Determine-Unit-EAS. The *Determine-Unit-EAS* operator determines the unit of measure for the quantity of work associated with an element activity. The operator performs four steps:

- Step 1.* If the unit of measure can be inherited from a parent schema in the element activity tree: exit.
- Step 2.* Build the name of the *Unit* KS (see p. 214) to be evaluated by concatenating the prefix *KS-Unit-of-Measure-* with the first two parts of the element activity code (division and broadscope codes, i.e., the *ea-code* slot value).
- Step 3.* Evaluate this KS and return the unit of measure for the activity.
- Step 4.* Store the unit of measure in the broadscope parent schema corresponding to the element activity (e.g., the broadscope parent of the schema *F00-EA-3-310-10-60-1* is *F00-EA-3-310*). Store the name of the *Unit* KS in the *why-unit* slot of the element activity.

Figure 6-15 shows the DOS that describes the *Determine-Unit-EAS* operator of CONSTRUCTION PLANEX. The operator uses the element activity code (*ea-code*) to identify the *Unit* KS to be evaluated and fills the slot *amount-unit*. The DOS indicates that the result is stored in the element activity schema, but the operator stores the result in the parent schema. For control purposes, this distinction is immaterial. Thus, the DOS does not differentiate between storing the unit of measure locally or in a parent schema of the element activity tree.

```
(defschema Determine-Unit-EAS
  (is-a          operator)
  (domain-type   ea)
  (application-object current-object)
  (input-objects current-object)
  (input-slots   ea-code)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object)
  (output-slots   amount-unit)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 6-15. Domain Operator Schema for the *Determine-Unit-EAS* Operator


```

(defschema Determine-Material-EAS
  (is-a operator)
  (domain-type ea)
  (application-object current-object)
  (input-objects current-object)
  (input-slots ea-code)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object)
  (output-slots material-package)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))

```

Figure 6-16. Domain Operator Schema for the *Determine-Material-EAS* Operator

Determine-Material-EAS. Figure 6-16 presents the DOS that describes the *Determine-Material-EAS* operator used to select the materials needed to perform an element activity. In a manner similar to the other element activity operators, the element activity code designation in the *ea-code* slot is used to identify the name of the KS to be evaluated. The operator fills the *material-package* slot of the element activity with the name of the materials needed for the activity.

The operator selects the material package in three steps:

- Step 1.* Build the name of the *Material* KS (see p. 214) to be evaluated by concatenating the prefix *KS-Material-* with the element activity code (division, broadscope and narrowscope codes, i.e., the *ea-code* slot value).
- Step 2.* Evaluate this KS to obtain the name of the selected material package.
- Step 3.* Store the name of the material package in the *material-package* slot of the element activity schema. Store the name of the *Material* KS in the *why-material* slot.

Create-PAS-for-EAS. Figure 6-17 shows the schema that describes the *Create-PAS-for-EAS* operator which is responsible for aggregating element activities into project activities. This DOS is very similar the DOS of the *Create-EAS* operator (see Figure 6-12). The operator uses the activity code (*ea-code*) to identify the *Project Activity* KS to be evaluated and links the element activity object to a project activity schema using the *ea-of-pa* slot. The other effects of the operator are unpredictable because the name of the project activity schema where results are stored is not known until the KS has been evaluated.

When the operator is applied to an element activity, five steps are performed:

- Step 1.* Build the name of the *Project Activity* KS (see p. 216) to be evaluated by concatenating the prefix *KS-Create-PA-* with the element activity

```
(defschema Create-PAS-for-EAS
  (is-a          operator)
  (domain-type   ea)
  (application-object current-object)
  (input-objects current-object)
  (input-slots    ea-code)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object <pa> <pa>)
  (output-slots    ea-of-pa is-a pa-has-eas)
  (output-bindings <pa> nil nil)
  (output-predictable yes no no)
  (output-effect-type fill fill fill))
```

Figure 6-17. Domain Operator Schema for the *Create-PAS-for-EAS* Operator

code (i.e., the *ea-code* slot which contains the division, broadscope group and narrowscope group codes).

- Step 2.* Evaluate this KS to obtain a pair (*pa-schema-name pa-name*) which specifies the project activity name and the project activity schema name to which the element activity is linked.
- Step 3.* If the project activity schema does not exist: create a new project activity schema and store the name of the project activity in this schema.
- Step 4.* Link the element activity to the project activity using the *ea-of-pa* relationship.
- Step 5.* Store the name of the *Project Activity* KS used in the *why-eas* slot of the element activity.

Get-Duration-EAS. The DOS that describes the preconditions and effects of the *Get-Duration-EAS* operator is shown in Figure 6-18. This operator estimates the duration of element activities using inherited information from the project activity schema to which the element activity is linked and stores the result in the *duration-ea* slot. The adjusted productivity of the crew assigned to the project activity must have been previously computed and stored in the *adj-productivity* slot of the project activity object.

The *Get-Duration-EAS* operator is algorithmic and does not require KS evaluation. The duration of the element activity is computed by dividing its total amount of work by the adjusted productivity of the crew allocated to the element activity. This productivity value is stored in the project activity schema to which the element activity is linked.

```

(defschema Get-Duration-EAS
  (is-a operator)
  (domain-type ea)
  (application-object current-object)
  (input-objects current-object <pa> current-object)
  (input-slots ea-of-pa adj-productivity
    amount-of-work-ea)
  (input-bindings <pa> nil nil)
  (input-cond-types filled filled filled)
  (output-objects current-object)
  (output-slots duration-ea)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))

```

Figure 6-18. Domain Operator Schema for the *Get-Duration-EAS* Operator

6.2.2.3 Project Activity Operators There are eight operators that manipulate individual project activity schemas in CONSTRUCTION PLANEX:

- *Create-PA-Tree* builds the tree of project activities;
- *Select-Technology-PAS* chooses crew types for project activities;
- *Compute-Amount-PAS* computes the quantity take-offs for the project activities;
- *Determine-Recommended-Duration-PAS* recommends an appropriate duration for project activities;
- *Get-Duration-PAS* determines how many crews to allocate to the activities and computes the normal and overtime hours of activities;
- *Get-Successors-PAS* establishes precedences among project activities;
- *Get-Lags-PAS* computes the leads and lags between consecutive project activities; and
- *Get-Cost-PAS* estimates the cost of the project activities.

In addition, the system includes two operators that are applied to the set of project activities as a whole:

- the *Floyd-Warshall* operator computes the earliest and latest start and finish times for the project activities; and
- the *Compute-NPV* operator determines the net present value of a project using scheduling and financial information.

These two operators do not have associated DOSs; the user invokes them directly. Both operators are complex and are applied to a complete set of project activities and thus they are impacted by most changes to the context. Since DOSs do not exist, these operators are not added to the *agenda* each time a new assertion is made. Thus, manual control limits when they are applied, which is particularly important for the computationally intense *Floyd-Warshall* operator.

```

(defschema Create-PA-Tree
  (is-a          operator)
  (domain-type   pa)
  (application-object current-object)
  (input-objects current-object)
  (input-slots   is-a)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object current-object)
  (output-slots   parent-pa pa-code)
  (output-bindings nil nil)
  (output-predictable yes yes)
  (output-effect-type fill fill))

```

Figure 6-19. Domain Operator Schema for the *Create-PA-Tree* Operator

Create-PA-Tree. The tree of project activities (as described in Section 6.2.1.3) is built by the *Create-PA-Tree* operator. The operator builds the tree bottom up, like the *Create-DE-Tree* and *Create-EA-Tree* operators, repeatedly applying three steps until the root of the tree is reached:

- Step 1.* Create the name of the immediate parent schema in the project activity tree by trimming the last identifier from the name of the daughter schema.
- Step 2.* If the parent schema does not exist: create it.
- Step 3.* Link the daughter project element activity schema to the parent project element activity schema.

Figure 6-19 shows the DOS that describes the *Create-PA-Tree* operator. The operator fills the slots *parent-pa* and *pa-code* of the project activities being linked into the tree using the name of the project activity schema.

Select-Technology-PAS. Technologies used to perform project activities are chosen by the *Select-Technology-PAS* operator. The DOS is shown in Figure 6-20. The operator uses the project activity code to identify which *Technology* KS to evaluate. It fills the *technology* slot of the project activity schema. This value is not stored locally in the project activity schema, but in a group technology object, as described in Section 6.1.2. Crew selection is a four-step process:

- Step 1.* Build the name of the *Technology* KS (see p. 218) to be evaluated by concatenating the prefix *KS-Technology-* with the first part of the project activity code (i.e., the first identifier from the value of the *pa-code* slot).
- Step 2.* Evaluate this KS and return the name of the most appropriate crew for performing the activity.

```
(defschema Select-Technology-PAS
  (is-a      operator)
  (domain-type pa)
  (application-object current-object)
  (input-objects  current-object)
  (input-slots    pa-code)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object)
  (output-slots    technology)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 6-20. Domain Operator Schema for the *Select-Technology-PAS* Operator

- Step 3.* If this crew type has not been stored in one of the *group-technology* objects of the context (their names are stored in the *is-a+inv* slot of the *group-technology* schema): create a new *group-technology* object.
- Step 4.* Link the project activity to the *group-technology* object describing the crew using the *parent-technology* relationship.

Compute-Amount-PAS. The *Compute-Amount-PAS* operator determines work quantities for project activities. This operator is algorithmic and does not require KS evaluation. It simply adds the quantity take-offs of the element activities which comprise a particular project activity and stores the result in the *amount-of-work-pa* slot of the project activity schema.

The DOS for the *Compute-Amount-PAS* operator of CONSTRUCTION PLANEX is shown in Figure 6-21. The operator inputs are the element activity work quantities (*amount-of-work-ea*). The operator has a single output, the result slot (*amount-of-work-pa*).

```
(defschema Compute-Amount-PAS
  (is-a      operator)
  (domain-type pa)
  (application-object current-object)
  (input-objects  current-object <eas>)
  (input-slots    pa-has-eas amount-of-work-ea)
  (input-bindings <eas> nil)
  (input-cond-types filled filled)
  (output-objects current-object)
  (output-slots    amount-of-work-pa)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 6-21. Domain Operator Schema for the *Compute-Amount-PAS* Operator

```

(defschema Determine-Recommended-Duration-PAS
  (is-a                operator)
  (domain-type         pa)
  (application-object  current-object)
  (input-objects       current-object current-object)
  (input-slots         pa-code amount-of-work-pa)
  (input-bindings      nil nil)
  (input-cond-types    filled filled)
  (output-objects      current-object)
  (output-slots        recommended-duration)
  (output-bindings     nil)
  (output-predictable  yes)
  (output-effect-type  fill))

```

Figure 6-22. Domain Operator Schema for the *Determine-Recommended-Duration-PAS* Operator

Determine-Recommended-Duration-PAS. Computation of all the recommended activity durations for project activities is provided by the operator *Determine-Recommended-Duration-PAS*. The operator uses the following procedure:

- Step 1.* Build the name of the *Duration* KS (see p. 218) to be evaluated by concatenating the prefix *KS-Dura-* with the project activity code (*pa-code*).
- Step 2.* Evaluate this KS and return the recommended project activity duration.
- Step 3.* Store the duration in the *recommend-duration* slot of the project activity schema and store the name of the *Duration* KS in the *why-duration* slot of the project activity.

The DOS that describes the *Determine-Recommended-Duration-PAS* operator is presented in Figure 6-22. This operator requires the *pa-code* slot as input to identify the KS to be evaluated. The quantity of work used to determine the duration comes from the *amount-of-work-pa* slot of the activity. The result is stored in the output slot *recommended-duration*.

Get-Duration-PAS. The *Get-Duration-PAS* operator uses the process outlined in Section 6.1.3 to estimate the duration of project activities and compute the distribution of normal and overtime hours. The operator is purely algorithmic and does not require the evaluation of any KS.

The DOS of this operator is shown in Figure 6-23. The operator uses as input the recommended duration of the activity (*recommended-duration*), the amount of work (*amount-of-work-pa*) and the standard productivity of the crew (*std-productivity*). It then fills the *duration* (estimated activity duration), *adj-productivity* (adjusted crew productivity), *normal-hours* (number of normal working hours), *overtime-hours* (number of overtime hours) and *number-crews* (number of crews allocated to the activity) slots of the project activity schema.

```

(defschema Get-Duration-PAS
  (is-a operator)
  (domain-type pa)
  (application-object current-object)
  (input-objects current-object <crew> current-object
                 current-object)
  (input-slots technology std-productivity
               amount-of-work-pa
               recommended-duration)
  (input-bindings <crew> nil nil nil)
  (input-cond-types filled filled filled filled)
  (output-objects current-object current-object
                 current-object current-object
                 current-object)
  (output-slots normal-hours overtime-hours duration
               number-crews adj-productivity)
  (output-bindings nil nil nil nil nil)
  (output-predictable yes yes yes yes yes)
  (output-effect-type fill fill fill fill fill))

```

Figure 6-23. Domain Operator Schema for the *Get-Duration-PAS* Operator

Get-Successors-PAS. The *Get-Successors-PAS* operator is used to determine the successor activities of a project activity. The operator performs three steps:

- Step 1.* Build the name of the *Successors* KS (see p. 219) to be evaluated by concatenating the prefix *KS-Succ-* with the project activity code (the *pa-code* slot value).
- Step 2.* Evaluate this KS and return a list of successors activities.
- Step 3.* Add each member of the list to the *successors* slot of the project activity and store the name of the evaluated KS in the *why-successors* slot.

The DOS for the *Get-Successors-PAS* operator is shown in Figure 6-24. It has one input and one output. The operator uses the code of the project activity (*pa-code*) to identify the KS to be evaluated, and stores the results of this evaluation in the *successors* slot of the project activity schema.

Get-Lags-PAS. The *Get-Lags-PAS* operator determines the leads and lags between consecutive project activities. This information is used to create the unified activity network model described in Section 6.1.4. This operator performs the following steps for each successor of a project activity:

- Step 1.* Build the name of the *Lag* KS (see p. 220) to be evaluated by concatenating the prefix *KS-Lag-* with the project activity code (*pa-code*) of the project activity and the project activity code (*pa-code*) of the successor activity.

```
(defschema Get-Successors-PAS
  (is-a          operator)
  (domain-type   pa)
  (application-object current-object)
  (input-objects current-object)
  (input-slots   pa-code)
  (input-bindings nil)
  (input-cond-types filled filled)
  (output-objects current-object)
  (output-slots   successors)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 6-24. Domain Operator Schema for the *Get-Successors-PAS* Operator

```
(defschema Get-Lags-PAS
  (is-a          operator)
  (domain-type   pa)
  (application-object current-object)
  (input-objects  current-object current-object <succs>)
  (input-slots    pa-code successors pa-code)
  (input-bindings nil <succs> nil)
  (input-cond-types filled filled filled)
  (output-objects current-object current-object
                  current-object current-object)
  (output-slots   succs link low-lags high-lags)
  (output-bindings nil nil nil nil)
  (output-predictable yes yes yes yes)
  (output-effect-type fill fill fill fill))
```

Figure 6-25. Domain Operator Schema for the *Get-Lags-PAS* Operator

- Step 2.* Evaluate this KS to obtain a list of pairs of lead or lag type and value (*lag-type lag-value*).
- Step 3.* Store the types of lags in the *link* slot of the project activity schema and their values in the *low-lags* slot. Store the name of the KS used in the *why-lags* slot of the project activity.
- Step 4.* Fill the *succs* slot with the name of a successor for each type of precedence link (a successor appears twice in this slot if both activities are linked using an SS and an FF link, but appears only once in the *successor* slot).
- Step 5.* Fill the *high-lags* slot with the upper bound of the lag for each link (assumed to be infinite when no restrictions are imposed).

The DOS that describes the *Get-Lags-PAS* operator is presented in Figure 6-25. It requires as input the project activity code (*pa-code*) of the activity to which the operator is applied and the project activity codes of its

successor activities. The result of its action is four *precedence* slots of the project activity are filled.

Get-Cost-PAS. Figure 6-26 shows the DOS for the *Get-Cost-PAS* operator used to compute costs for project activities. The operator requires as input the duration of the project activity (*normal-hours*, *overtime-hours*), the hourly cost of the crew assigned to the project activity (*normal-cost*, *overtime-cost*), and the material cost of the material package used by the activity (*mat-unit-cost*). The operator fills the *cost* slots of the project activity with the resultant values.

The *Get-Cost-PAS* operator is algorithmic and computes both crew and material costs as follows:

- Step 1.* Compute crew costs of the project activity by multiplying the normal and overtime hours by the corresponding crew unit costs.
- Step 2.* Compute material costs of the project activity by multiplying the work quantity by the cost of the material package.
- Step 3.* Sum the crew and material costs for the project activity to obtain totals and daily totals.

```
(defschema Get-Cost-PAS
  (is-a          operator)
  (domain-type   pa)
  (application-object current-object)
  (input-objects current-object current-object
                 current-object current-object <eas>
                 <crew> <crew> <material>)
  (input-slots   has-eas normal-hours overtime-hours
                 technology material-package
                 normal-cost overtime-cost
                 mat-unit-cost)
  (input-bindings <eas> nil nil <crew> <material> nil
                 nil nil)
  (input-cond-types filled filled filled filled filled
                    filled filled filled)
  (output-objects current-object current-object
                  current-object current-object
                  current-object current-object)
  (output-slots   overall-total-cost
                  overall-cost-per-day
                  total-cost-crew cost-crew-per-day
                  total-cost-materials
                  cost-materials-per-day)
  (output-bindings nil nil nil nil nil nil)
  (output-predictable yes yes yes yes yes yes)
  (output-effect-type fill fill fill fill fill fill))
```

Figure 6-26. Domain Operator Schema for the *Get-Cost-PAS* Operator

Compute-NPV. The *Compute-NPV* operator is used to compute the net present value of the project. The discounted project costs are computed using the contractor's minimum attractive rate of return. All costs are assumed to be incurred at the start of the activity using an earliest start schedule. Costs are computed on a monthly basis.

Floyd-Warshall. The *Floyd-Warshall* operator is used to compute a project schedule. It uses the unified activity network model description of the project activities stored in the *precedence* and *duration* slots of the project activity schema. It computes earliest and latest start and finish time for all activities using the Floyd-Warshall scheduling algorithm. These results are stored in the *scheduling* slots of the project activities.

6.2.3 Knowledge Sources

The knowledge base of CONSTRUCTION PLANEX consists of the KSs that store the knowledge used by the domain operators to generate a construction plan for a building. As noted previously, the knowledge in these KSs is limited to that for the planning of the excavation and erection of concrete or steel buildings. The current construction knowledge yields a fast-track schedule. The knowledge comes from several sources: (1) an experienced construction planner [4]; (2) publications on building estimating procedures [19, 105]; (3) publications on construction methods [76]; and (4) building cost data [71]. The following types of knowledge sources are present in CONSTRUCTION PLANEX:

- *Element Activity* KSs describe the set of activities required to construct a design element;
- *Amount* KSs specify formulas used when computing the quantity of work for each element activity;
- *Unit* KSs indicate the default unit of measure of the work quantities for each type of element activity;
- *Material* KSs specify the set of materials used in performing an element activity;
- *Project Activity* KSs specify the name of the project activity associated with a particular element activity;
- *Technology* KSs recommend appropriate crews for constructing project activities;
- *Duration* KSs compute desirable durations for project activities;
- *Successor* KSs specify the names of the successors of particular project activities; and
- *Lag* KSs are used to determine the types and values of leads and lags between two consecutive project activities.

Each type of KS is described below and illustrated with an example.

Knowledge Sources for Element Activity Creation. Determining the set of element activities used to construct each design element is the responsibility of the *Element Activity Creation* KSs. These KSs are used by the *Create-EAS* operator (see p. 196) to generate element activity schemas.

Element activity creation KSs are organized according to the same coding system used to organize design element schemas; i.e., organized by type of design element and type of material (*de-code*). The name of an element activity KS is of the form: *KS-Create-EA-⟨type-de⟩-⟨type-material⟩*. For example, the KS that generates the codes and names of the element activities required to erect a steel beam (i.e., design element type "81" and material type "2") is *KS-Create-EA-81-2*. Thus, the domain operator responsible for element activity creation can readily identify which KS to evaluate for a given type of design element.

An example of an element activity creation KS for column footings is shown in Figure 6-27. This KS is similar to the example used in Section 4.2 to describe the knowledge representation scheme of PLANEX. It returns the names and codes of the element activities required to build a column footing. The firing type of this KS is "all": all rules are fired sequentially. The KS is applied to a design element (*current-object*). The *cond-objects* slot indicates that the information needed to evaluate the KS is stored in the schema of the design element being analyzed (the *current-object*) and the *soil-info* object. The KS of Figure 6-27 contains four conditions:

- the first condition binds the value of the *root-code* slot of the design element object to the binding variable *⟨root⟩*;
- the second condition determines if the *type-de* slot of the design element is a concrete footing (i.e., has value "60");
- the third condition binds the value of the *number-de* slot (e.g., which design element group) of the design element to the variable *⟨number⟩*; and
- the fourth condition tests if the *possible-use* slot of the *soil-info* object has a value "backfill".

The KS has three rules:

- Rule 1.* All concrete column footings require excavation, building formwork, placing reinforcing, pouring concrete and stripping forms.
- Rule 2.* If the soil is appropriate for backfill, constructing the footing requires the excavated material to be piled up for later use in backfill.
- Rule 3.* If the soil is not appropriate for backfill, the contractor has to dispose of the excavated material and borrow backfill material from a different source.

As described in Section 4.2.3, conditions 1 and 3 are binding conditions and are always true. They are used to compute codes and names of the element

```

(defschema KS-Create-EA-60-1
  (is-a      ks)
  (ks-name   KS-Create-EA-60-1)
  (ks-type   all)
  (cond-objects current-object current-object current-object
                soil-info)
  (conditions (= root-code <root>)
              (= type-de 60)
              (= number-de <number>)
              (= possible-use backfill))
  (lhs-rules (T T T I)
             (T T T T)
             (T T T F))
  (rhs-rules (X I I I X X X X)
             (I I X I I I I I)
             (I X I X I I I I))
  (actions  (<root>-EA-2-220-10-60-1-<number>
            excavate-column-footings-<number>)
            (<root>-EA-2-225-10-60-1-<number>
            dispose-excavation-column-footings-<number>)
            (<root>-EA-2-225-20-60-1-<number>
            pile-up-excavation-column-footings-<number>)
            (<root>-EA-2-220-40-60-1-<number>
            borrow-material-column-footings-<number>)
            (<root>-EA-3-110-10-60-1-<number>
            place-forms-column-footings-<number>)
            (<root>-EA-3-210-00-60-1-<number>
            reinforce-column-footings-<number>)
            (<root>-EA-3-310-10-60-1-<number>
            pour-concrete-column-footings-<number>)
            (<root>-EA-3-110-20-60-1-<number>
            remove-forms-column-footings-<number>)))

```

Figure 6-27. Example of a KS for Element Activity Creation

activity schemas. If the root-code of the design element is *F00* and its group number is "1", firing the first rule produces the list of actions:

```

(F00-EA-2-220-10-60-1-1 excavate-column-footings-1)
(F00-EA-3-110-10-60-1-1 place-forms-column-footings-1)
(F00-EA-3-210-00-60-1-1 reinforce-column-footings-1)
(F00-EA-3-310-10-60-1-1 pour-concrete-column-footings-1)
(F00-EA-3-110-20-60-1-1 remove-forms-column-footings-1)

```

which represents the set of activities required to construct any concrete column footing. This list is used by the *Create-EAS* operator to create the corresponding element activity schemas.

```
(defschema KS-Amount-2-225-10
  (is-a      ks)
  (ks-type   first)
  (cond-objects current-object soil-info)
  (conditions (= type-de 60)
               (= stability-angle 90.0))
  (lhs-rules (T T)
              (T F))
  (rhs-rules (X I)
              (I X))
  (actions   formula-04 formula-05))
```

Figure 6-28. Example of a KS to Compute Amounts of Work

```
(setq formula-04 '(
  (((x1-dimension + 4) * (y1-dimension + 4) *
    (abs zg-coordinate) + (abs z1-dimension))) / 27.0)
  * KS-swell-factor))
```

Figure 6-29. Example of a Formula to Compute Amounts of Work

Knowledge Sources for Computing Amounts of Work. Amount KSs are used to select formulas for computing work quantities for element activities. The KS is used by the *Compute-Amount-EAS* operator (see p. 199). The KS naming convention uses the same coding system used for element activities and takes the form: *KS-Amount-(division)-(broadscope)-(narrowscope)*, where the division, broadscope and narrowscope numbers correspond to the first three parts of the element activity code.

Figure 6-28 shows one of the KSs that CONSTRUCTION PLANEX uses to compute the quantity of work for an excavation element activity. The KS returns the names of formulas which are evaluated using the geometric description of the corresponding design elements to yield the work quantity. The KS returns one of two alternative formulas for computing the amount of material to be disposed of when excavating a column footing. The KS has two rules:

- Rule 1.* For a column footing design element (*type-de* has a value equal to "60") and soil stability angle of ninety degrees (i.e., vertical excavation walls will not collapse), "formula-04" is used. This is the formula presented in Figure 6-29; it expresses the excavation volume in terms of the dimensions of the column footing.
- Rule 2.* For a column footing design element and soil stability angle not ninety degrees (a slope is required), the more complex "formula-05" is used.

The formulas are procedural code elements of the knowledge base. The formulas might include names of KSs that are evaluated before the final quantity is computed. For example, the formula of Figure 6-29 includes a term called

```
(defschema KS-Unit-of-Measure-3-310
  (is-a      ks)
  (ks-name   KS-Unit-of-Measure-3-310)
  (ks-type   first)
  (cond-objects current-object)
  (conditions (= narrowscope 10))
  (lhs-rules (T))
  (rhs-rules (X))
  (actions   cu-yd))
```

Figure 6-30. Example of a KS to Determine Units of Measure

KS-swell-factor that represents the swell factor of the soil. Any term of the formula that has an *is-a+inv* slot indicating it is a KS is evaluated by the KSE and the result of the evaluation is substituted into the formula for the corresponding term.

Knowledge Sources for Units of Measure. Unit KSs are utilized by the *Determine-Unit-EAS* domain operator to determine the unit of measure for quantity take-offs of element activities (see p. 200). These KSs are indexed at the broadscope level of the element activity hierarchy. The KS name is of the form: *KS-Unit-of-Measure-(division)-(broadscope)*, where the division and broadscope group numbers correspond to the first two parts of the *ea-code-slot* slot value. The KSs return units of measure for each of the corresponding narrow-scope codes.

An example of a KS to determine the units for work quantities is shown in Figure 6-30. The KS contains one rule that indicates that for the activities whose *ea-code* starts with "3-310-10", the amount of work is expressed in cubic yards.

In the current version of CONSTRUCTION PLANEX, unit conversions are not performed. Units are simply recorded and the system assumes that all quantities are expressed in consistent units. For example, the formula for quantity take-off in Figure 6-29 assumes that all dimensions are expressed in feet and uses the factor "27.0" to convert the volume to cubic yards.

Knowledge Sources for Materials Selection. Similarly to the selection of technologies (described in Section 6.1.2), materials used in the construction of an element activity are defined in terms of material packages. Each material package is a group of individual materials such as cement, aggregates and water. Material package selection for an element activity is performed by the *Determine-Material-EAS* operator (see p. 201).

The organization of the *Material* KSs is identical to that used for *Amount* KSs: structured according to element activity code. Names are of the form *KS-Material-(division)-(broadscope)-(narrowscope)*.

```
(defschema KS-Material-3-110-10
  (is-a      ks)
  (ks-name   KS-Material-3-110-10)
  (ks-type   first)
  (cond-objects current-object specif-info)
  (conditions (= type-de <type>)
               (= formwork-<type> special))

  (lhs-rules (T F)
              (T T))
  (rhs-rules (X I)
              (I X))
  (actions   formwork-<type>-normal
              formwork-<type>-special))
```

Figure 6-31. Example of a KS for Material Package Selection

```
(defschema formwork-81-special
;
; Material package for beams
; Based on 1 use. Based on 12 inch beams. Note 33 means
; This package corresponds to the case when
; formwork-beams is special in the specif-info frame
;
  (is-a      material-package)
  (component-names ((1.1 5/8-plyform); sq-ft of 5/8 plyform
                    (2.1 lumber) ; board feet of lumber
                    (allow accessories))); other accessories
  (material-unit sq-ft)
  (mat-unit-cost 1.62))
```

Figure 6-32. Example of a Material Package

Figure 6-31 shows the KS that is used to select materials for element activities grouped below the *3-110-10* narrowscope level of the element activity tree. To evaluate this KS for a particular element activity, the system uses information stored in the schema titled *specif-info*, which contains project specification data not associated with individual design elements. The KS has two rules:

- Rule 1.* If the formwork type of the design element associated with the element activity is "special" (e.g., the *formwork-<type>* slot of the *specif-info* has the value "special" where *<type>* is the design element type: *type-de*), the name of the material package is "formwork-81-special".
- Rule 2.* If the formwork type is not "special", the name of the material package is "formwork-81-normal".

The material package *formwork-81-special* is shown in Figure 6-32. This schema contains information describing the individual components of the pack-

```

(defschema KS-PA-3-110-10
  (is-a      ks)
  (ks-name   KS-PA-3-110-10)
  (ks-type   first)
  (cond-objects current-object current-object
                 current-object current-object
                 current-object)
  (conditions (= type-de 60)      ; column footings
               (= type-de 65)      ; columns
               (= type-de 80)      ; slabs
               (= type-de 81)      ; beams
               (= root-code <root>)) ; bind to location
  (lhs-rules (T F F F T)
              (F T F F T)
              (F F T F T)
              (F F F T T))
  (rhs-rules (X I I I)
              (I X I I)
              (I I X I)
              (I I I X))
  (actions (<root>-PA-20-60 formwork-foundation-<root>)
            (<root>-PA-20-65 formwork-columns-<root>)
            (<root>-PA-20-80 formwork-slab-<root>)
            (<root>-PA-20-80 formwork-slab-<root>)))

```

Figure 6-33. Example of a KS for Project Activity Creation

age and the average cost of materials per unit of activity work (in the specified units).

Knowledge Sources for Project Activity Creation. The *Project Activity* KSs are used by the *Create-PAS-for-EAS* operator (see p. 201) to define which element activities are aggregated into project activities. The KSs are organized according to the same coding system used for element activities. The KS name is of the form: *KS-PA-⟨division⟩-⟨broadscope⟩-⟨narrowscope⟩*, where the division, broadscope group and narrowscope group codes correspond to the first three parts of the element activity code of the activity to which the operator is applied.

An example of a project activity creation KS is shown in Figure 6-33. The *ks-type* of this KS is "first", indicating that only one rule should be fired. The KS contains four rules:

- Rule 1.* Formwork activities of column footings are aggregated into a project activity that groups all formwork of foundation elements.
- Rule 2.* Formwork activities of columns are aggregated into a project activity that groups all formwork for the columns of a particular floor.
- Rule 3.* Formwork activities of slabs are aggregated into a project activity that groups all formwork activities associated with slab and beam elements of a particular floor.


```

(defschema KS-Tech-20
;;
;; This KS is used to determine the crew type of formwork
;; activities.
;; The function "check-elements-foundation" returns true
;; if all foundation elements are column footings and false
;; otherwise
;;
      (is-a      ks)
      (ks-name   KS-Tech-20)
      (ks-type   first)
      (cond-objects current-object function none current-object)
      (conditions (= pa-code 20-60)
                  (= (check-elements-foundation
                     current-object) t)
                  (= pa-code 20-80))
      (lhs-rules  (T T F)
                  (F I F)
                  (F I T))
      (rhs-rules  (X I)
                  (X I)
                  (I X))
      (actions    crew-formwork-05
                  crew-formwork-06))

```

Figure 6-34. Example of a KS for Technology Selection

Rule 4. Formwork activities of beams are aggregated into a project activity that groups all formwork activities associated with slab and beam elements of a particular floor.

Knowledge Sources for Technology Selection. As described in Section 6.1.2, CONSTRUCTION PLANEX selects construction technologies and assigns crews to project activities. The appropriate crew type for a project activity is identified by the *Select-Technology-PAS* operator (see p. 204) using a *Technology* KS. These are organized by project activity code, and the KS name is based on the first part of the project activity code. A KS titled *KS-Technology-30* is used to determine the crew type for project activities having a code starting with the number "30" (formwork stripping for structural elements).

An example of a *Technology* KS is shown in Figure 6-34. This KS returns the name of a crew used in formwork placement project activities. There are three rules:

- Rule 1.* If the project activity is foundation column form placement, use crew "crew-formwork-05".
- Rule 2.* If the project activity is not foundation column form placement or slab form placement, use crew "crew-formwork-05".

```

(defschema KS-Dura-30-80
  (is-a      ks)
  (ks-name   KS-Dura-30-80)
  (ks-type   first)
  (cond-objects current-object
                current-object
                current-object)
  (conditions (<= amount-of-work-pa 6400)
              (<= amount-of-work-pa 12800)
              (<= amount-of-work-pa 19200))
  (lhs-rules (T I I)
             (F T I)
             (I F T)
             (I I F))
  (rhs-rules (X I I I)
             (I X I I)
             (I I X I)
             (I I I X))
  (actions   5 7 10 15))

```

Figure 6-35. Example of a KS to Recommend Durations

Rule 3. If the project activity is slab formwork placement use, crew “crew-formwork-06”.

An example of a crew schema was presented in Figure 6-1.

Knowledge Sources for Activity Durations. The domain operator *Determine-Recommend-Duration-PAS* uses *Duration* KSs to compute the recommended durations for project activities (see p. 206). CONSTRUCTION PLANEX uses the results provided by *Duration* KSs to select the number of crews to allocate to the project activities (see Section 6.1.2). Following this computation, the estimated durations of the activities and the distribution of normal and overtime hours are calculated using the procedure outlined in Section 6.1.3.

The name of a *Duration* KS is of the form: *KS-Dura-⟨pa-code⟩*, where *⟨pa-code⟩* is the code of the project activity for which the duration is being computed. For example, the recommended duration for the activity representing column form placement (its project activity code is “20-65”) is determined by the KS titled *KS-Dura-20-65*.

An example of a *Duration* KS is presented in Figure 6-35. This KS returns the recommended duration of an activity in days for various values of the work quantity (*amount-of-work-pa* slot) of the project activity. The KS contains four rules:

- Rule 1.* If the quantity of work is less than 6400, the recommended duration is 5 days.
- Rule 2.* If the quantity of work is greater than or equal to 6400 but less than 12800, the recommended duration is 7 days.
- Rule 3.* If the quantity of work is greater than or equal to 12800 but less than 19200, the recommended duration is 10 days.
- Rule 4.* If the quantity of work is greater than or equal to 19200, the recommended duration is 15 days.

Knowledge Sources for Successor Identification. Successor KSs are used to generate the successor activities of a project activity. These KSs are organized by project activity (i.e., the name of the KS is of the form *KS-Succ-(pa-code)*). They are used by the *Get-Successors-PAS* operator (see p. 207).

Figure 6-36 shows a KS that is used to determine the successors of a concrete pouring activity for column footings (identified with code "50-60"). The *ks-type* of this KS is "all", indicating that all applicable rules should be fired to generate all possible successors. Conditions check for the existence of successors using the *Schemap* function. This function takes the name of a schema as an argument and returns "true" if the schema exists. It is used to insure that a successor is generated only once. The KS has two rules:

- Rule 1.* If it does not exist, generate a successor activity for placing column footing forms (project activity code "20-65").
- Rule 2.* If it does not exist, generate a successor activity for stripping forms from the column footings (project activity code "30-60").

```
(defschema KS-Succ-50-60
  (is-a      ks)
  (ks-name   KS-Succ-50-60)
  (ks-type   all)
  (cond-objects current-object function function)
  (conditions
    (= root-code <root>)
    (= (schemap <root>-PA-30-60) t)
    (= (schemap <root>-PA-20-65) t))
  (lhs-rules (T T I)
             (T I T))
  (rhs-rules (X I)
             (I X))
  (actions   <root>-PA-30-60
             <root>-PA-20-65))
```

Figure 6-36. Example of a KS for Successor Identification

```

(defschema KS-Lag-40-60-to-40-65
  (is-a      ks)
  (ks-type   first)
  (cond-objects current-object function function function)
  (conditions
    (= root-code <root>)
    (= (get-eas-of-pa '<root>-PA-40-60)
       <list-curr>)
    (= (get-eas-of-pa '<root>-PA-40-65)
       <list-succ>))
  (lhs-rules (T T T))
  (rhs-rules (X X))
  (actions
    (SS (get-smallest-slot '<list-curr>'
                          'duration-ea))
    (FF (get-smallest-slot '<list-succ>'
                          'duration-ea))))

```

Figure 6-37. Example of a KS to Determine Lags

Knowledge Sources for Lag Determination. The *Get-Lags-PAS* operator (see p. 207) uses a *Lags* KS to identify the leads and lags between consecutive project activities. These KSs are organized according to the project activity codes for both of the activities. Thus, the form of a *Lags* KS name is *KS-Lag-⟨pa-code-1⟩-⟨pa-code-2⟩*, where *pa-code-1* is the project activity code of one activity and *pa-code-2* is the code of a successor project activity.

Figure 6-37 shows one of the KSs used in CONSTRUCTION PLANEX to determine lags between two project activities. This KS is used to compute the lag from a column footing reinforcing activity (project activity code "40-60") to a column reinforcing activity (project activity code "40-65"). The KS has one rule which computes both a Start-to-Start (SS) and a Finish-to-Finish (FF) precedence. The second and third conditions retrieve the sets of element activities which comprise the two project activities. These activity sets are used as arguments of the function *Get-Smallest-Slot* to determine the shortest duration element activity in each set. Then these durations are used to generate two lags:

1. The SS lag indicates that column reinforcing cannot start until the shortest duration footing reinforcing activity has been completed; and
2. The FF lag indicates that column reinforcing must finish at least *D* days after completing the footing reinforcing activity, where *D* is the duration of the shortest of the column reinforcing activities.

6.2.4 User Interface Mechanisms

In Section 4.4, some of the user interface mechanisms of CONSTRUCTION PLANEX were used to illustrate the interface components that may be incorporated in applications of the PLANEX architecture. CONSTRUCTION PLANEX interaction mechanisms include:

- the GANTT *Interactive Scheduler* which provides an interactive graphical display of the project schedule;
- the REPORT GENERATOR which outputs a variety of tabular reports detailing planning data;
- the CONTROL PANEL which provides mechanisms to alter the *agenda* and invoke control operators;
- interrogatives such as the questions issued by the *Get-Duration-PAS* operator (illustrated in Section 4.4.5);
- explanations of the problem solving process;
- passive graphical displays of results; and
- menus used to control the problem-solving process and access the mechanisms listed above.

This section describes some of the user interface mechanisms of the CONSTRUCTION PLANEX system such as menus, passive output graphics, output reports and explanations.

6.2.4.1 Menus Menus provide the mechanism to control the execution of domain and control operators. In CONSTRUCTION PLANEX, menus are used to:

- select an operator to execute;
- specify the context objects to which an operator is applied;
- set values for global variables; or
- answer a question that has a predefined, enumerated set of answers.

Figure 6-38 shows the structure of the menus used in CONSTRUCTION PLANEX. The menus form a hierarchy from left to right. Menus at the left of the figure are used to access menus at the right. For example, when the user chooses the "Change Values" option of the *Top* menu, the *Change* menu is displayed. Selecting a menu item either causes a corresponding submenu to be displayed or invokes the associated operator.

Top Menu. The *Top* menu is the root menu of the menu control structure with options to access all of the other portions of the menu interface. It is displayed when CONSTRUCTION PLANEX starts. This menu includes the following item:

- "Perform Operations" which accesses the *Operations* menu;
 - "Display Information or Results" which accesses the *Display* menu;
 - "Explain Results" which accesses the *Explain* menu;
 - "Change Values" which accesses the *Change* menu;
 - "Print Reports" which accesses the *Report* menu; and
 - "Exit" which terminates CONSTRUCTION PLANEX.
-

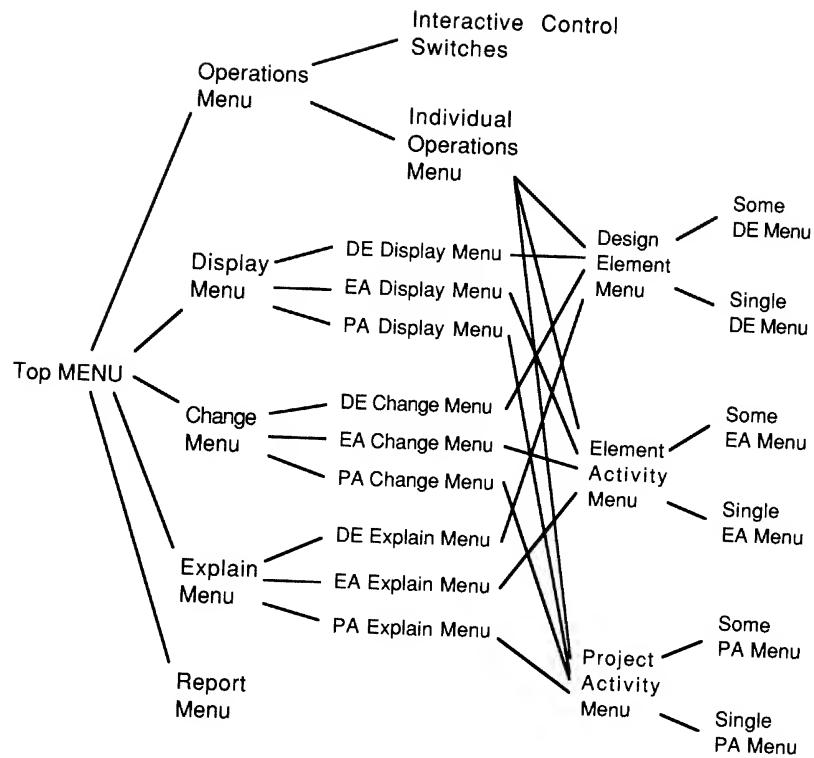


Figure 6-38. Menus of the CONSTRUCTION PLANEX System

Operations Menu. The *Operations* menu lets the user invoke a set of high-level control procedures. It includes the following items:

- "Translate a File from the Input Generator" which translates data from the INPUT GENERATOR [116] (see Section 4.4.2) into design element schemas;
- "Load a Building File" which inputs a file containing the design element schemas describing a building;
- "Complete Forward Pass" which sequentially applies a predefined set of operators to generate an initial project plan (this sequence of operators is described in Section 6.3.2);
- "Individual Operations" which accesses the *Individual Operations* menu;
- "Prepare File for Animation" which creates an output file for the ANIMATOR to display the construction sequence;
- "Interactive Control Panel" which is used to access the CONTROL PANEL (CP) (described in Section 4.4.3);

- “Modify Switches for Control Operators” which lets the user change the values of control variables via the *Control Switches* menu; and
- “Exit” which returns to the *Top* menu.

Context Object Operator Menus. Three menus, *Display*, *Change* and *Explain*, are used to access context objects. The user specifies the type of object requested for information display, change or explanation. Each of these menus has the following options:

- “Design Elements” which accesses information stored in design element schemas;
- “Element Activities” which accesses information stored in element activity schemas;
- “Project Activities” which accesses information stored in project activity schemas; and
- “Exit” which returns to the *Top* menu.

Each of the first three items corresponds to a *Context Object* submenu which lets the user select the object to be displayed, changed or for which an explanation is to be given.

Report Menu. The *Report* menu displays the names of the output reports that may be produced by CONSTRUCTION PLANEX. These reports are prepared by the REPORT GENERATOR, as described in Section 4.4.4. The names of available reports are stored in the *is-a+inv* slot of the *report* schema. CONSTRUCTION PLANEX retrieves these names and dynamically generates items on the *Report* menu during execution. Selecting a report name from the menu causes the corresponding report to be generated. CONSTRUCTION PLANEX reports are described in Section 6.2.4.3.

Control Switches Menu. The *Control Switches* menu provides the mechanism to set global variables which control the overall problem-solving behavior. Switches may be toggled on or off. The menu includes the following items:

- “Operator Preconditions” which controls whether unsatisfied operator preconditions are recorded as goals in the *goals* slot of the *agenda*;
 - “Operator Effects” which controls whether operator effects are recorded as changes in the *context-changes* slot of the *agenda*;
 - “Forward Pass” which specifies whether the number of questions presented to the user during the automated forward pass operation (see Section 6.3.2) should be minimized; and
 - “Exit” which returns to the *Operations* menu.
-

Individual Operations Menu. The *Individual Operations* menu provides a mechanism to invoke any of the CONSTRUCTION PLANEX domain operators, to enter the GANTT interactive scheduler or to display output graphics. This menu has the following options:

- “Create Tree of Design Elements” which executes the *Create-DE-Tree* operator;
- “Create Element Activities” which executes the *Create-EAS* operator;
- “Create Tree of Element Activities” which executes the *Create-EA-Tree* operator;
- “Determine Units of Measure” which executes the *Determine-Unit-EAS* operator;
- “Compute Amounts of Work for Element Activities” which executes the *Compute-Amount-EAS* operator;
- “Determine Material Packages” which executes the *Determine-Material-EAS* operator;
- “Create Project Activities” which executes the *Create-PAS-for-EAS* operator;
- “Create Tree of Project Activities” which executes the *Create-PA-Tree* operator;
- “Select Technology for Project Activities” which executes the *Select-Technology-PAS* operator;
- “Compute Amounts of Work for Project Activities” which executes the *Compute-Amount-PAS* operator;
- “Determine Recommended Durations for Project Activities” which executes the *Determine-Recommended-Duration-PAS* operator;
- “Compute Durations for Project Activities” which executes the *Get-Duration-PAS* operator;
- “Compute Durations for Element Activities” which executes the *Get-Duration-EAS* operator;
- “Determine Successors of Project Activities” which executes the *Get-Successor-PAS* operator;
- “Determine Lags among Project Activities” which executes the *Get-Lags-PAS* operator;
- “Estimate Cost of Project Activities” which executes the *Get-Cost-PAS* operator;
- “Compute the NPV of a Project” which executes the *Compute-NPV* operator;
- “Activity on Node Diagram” which displays an *Activity-On-Node* diagram of the project activity network;
- “Interactive Scheduling” which initiates GANTT, the interactive scheduler of CONSTRUCTION PLANEX;
- “Display Daily Project Cost Curve” which displays an X-Y plot of aggregated daily activity costs;

- “Display Cumulative Project Cost Curve” which displays an X-Y plot of aggregated cumulative activity costs; and
- “Exit” which returns to the *Operations* menu.

Context Object Menus. The *Design Element*, *Element Activity*, and *Project Activity* menus allow the user to specify the object to which an operator is applied. Whenever a *Display*, *Change*, *Explain* or *Individual Operations* menu item is selected, the appropriate *Design Element*, *Element Activity* or *Project Activity* submenu is displayed. For example, if the user invokes the *Compute-Amount-PAS* operator by selecting the “Compute Amounts of Work for Project Activities” option from the *Individual Operations* menu, the system displays the *Project Activity* menu for selecting the activities to which the operator will be applied. Each *Context Object* menu has a similar structure:

- “For all <xxxx>” specifies that the operator will be applied to all <xxxx>, where <xxxx> is either “Project Activity”, “Element Activity” or “Design Element”;
- “For some <xxxx>” specifies that the operator will be applied to some <xxxx>;
- “For a single <xxxx>” specifies that the operator will be applied to only one <xxxx>; and
- “Exit” returns to the previous menu.

If the “Some” option is selected, the user then chooses the objects by specifying their location and type. If the “Single” option is selected, the list of corresponding context objects is displayed and the user chooses one item from the list.

6.2.4.2 Output Graphics In addition to the interactive graphics of GANTT (see p. 126), CONSTRUCTION PLANEX includes the following passive output displays:

- the *Activity-On-Node* (AON) diagram displays the project activity network with some scheduling information;
- the *Cost Curve* displays an X-Y plot of total activity cash flow versus time;
- the *Cumulative Cost Curve* displays an X-Y plot of cumulative cash flows of project activities versus time; and
- an animation program, called ANIMATOR, provides a graphical simulation of the construction process.

Figure 6-39 illustrates the form of the AON representation of a project plan. In this diagram, activities are represented as boxes. Activities on the left precede those activities on the right. Each box displays activity schedule results: earliest and latest start and finish times. Additional specific activity information (e.g., the complete project activity schema) is displayed if the user points to a project box and clicks a mouse button.

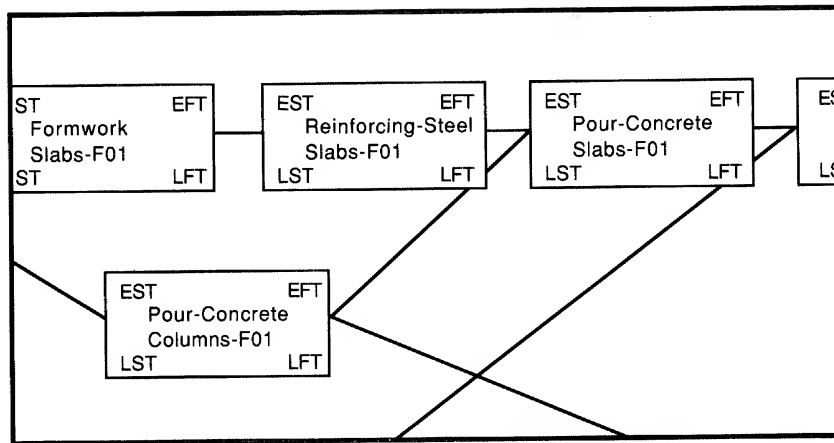


Figure 6-39. Illustration of an Activity-On-Arrow Diagram Display

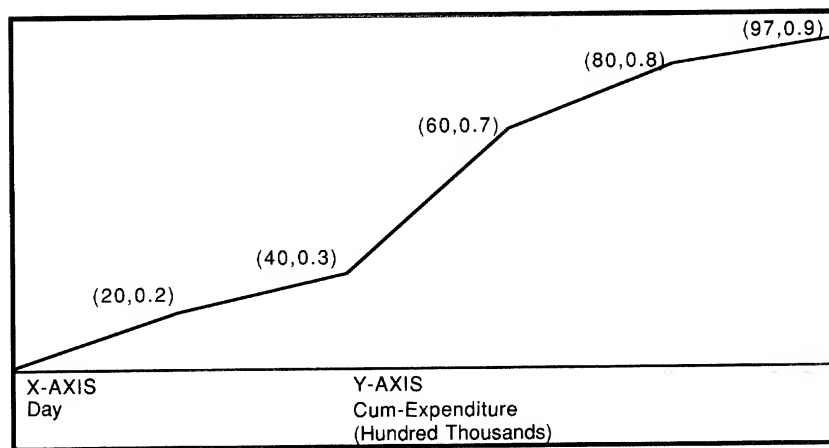


Figure 6-40. Illustration of the Cumulative Cost Curve Display

Figure 6-40 illustrates the display of the cumulative cost curve of a project plan. The Y-axis value corresponds to the accumulated project activity costs and the X-axis value is the project time-line. This cumulative cost curve is computed ignoring inflation and assuming that there is no time value of money (the minimum attractive rate of return of the contractor is zero). A more detailed cost computation can be obtained by invoking the *Compute-NPV* operator.

ANIMATOR [116] simulates the manner in which a building is constructed using 3-D output. The input to this program is a file created by CONSTRUCTION

PLANEX with information on the starting and ending times for each construction activity. ANIMATOR displays building components incrementally using different colors to represent different construction activities.

6.2.4.3 Output Reports CONSTRUCTION PLANEX produces a variety of reports containing information about the planning process. The user obtains reports by choosing the "Print Reports" option of the *Top* menu and by selecting one of the available report formats displayed in the *Report Menu*. Nonstandard reports may be produced by creating the appropriate report format schemas (see Section 4.4.4).

Figures 6-41 through 6-43 illustrate some of the reports that may be obtained:

- Figure 6-41 shows a schedule data report. It includes project activities whose earliest-start-time (EST) is less than 100 days. In this report, activities are sorted with respect to their EST.
- The report in Figure 6-42 shows the crew types for the project activities located on the first floor of the building (floor "F01"). In this report, activities are sorted by name. The report shows that the formwork activities for the columns (code "20-65") use a different crew ("crew-formwork-05") than the corresponding activities for slab elements (code "20-80", crew "crew-formwork-06").
- The report in Figure 6-43 shows how project activities contribute to the total project cost. In this report, activities are sorted by name and only those activities with a percent cost greater than 4% are printed.

CODE	LOCATION	DURATION	EST	EFT	LST	LFT
10-60	F00	35.0	0.00	35.00	0.00	35.00
20-60	F00	7.0	27.94	35.54	27.94	35.54
40-60	F00	11.0	28.48	39.48	28.48	39.85
40-65	F00	14.0	29.49	43.49	29.49	43.49
15-60	F00	2.0	32.70	35.61	184.26	186.26
50-60	F00	4.0	43.49	47.49	43.49	47.49
20-65	F00	12.0	44.49	56.49	44.49	56.49
30-60	F00	1.0	46.13	47.56	185.26	186.26
50-65	F00	3.0	56.49	59.49	56.49	59.60
20-80	F00	42.0	56.55	98.55	56.55	98.55
30-65	F00	3.0	57.49	60.49	183.26	186.26
17-60	F00	27.0	60.49	87.49	186.26	213.26
40-80	F00	26.0	73.64	99.82	73.82	99.82
50-80	F00	2.0	99.82	101.82	99.82	101.96

Figure 6-41. Example of a Scheduling Report

CODE	LOCATION	AMOUNT	CREW	NO.	DUR.
20-65	F01	5040.00	CREW-FORMWORK-05	6.0	8.0
20-80	F01	20261.00	CREW-FORMWORK-06	6.0	28.0
50-65	F01	130.67	CREW-POUR-CONCRETE-05	1.0	2.0
50-80	F01	504.14	CREW-POUR-CONCRETE-06	3.0	1.0
40-65	F01	27227.34	CREW-RE-STEEL-05	4.0	12.0
40-80	F01	79328.57	CREW-RE-STEEL-05	5.0	28.0
30-65	F01	5040.00	CREW-REMOVE-FORMS-06	4.0	3.0
30-80	F01	20261.00	CREW-REMOVE-FORMS-06	4.0	12.0

Figure 6-42. Example of a Report with Crew Type, Amount of Work and Duration

NAME	COST	PERCENT
BACKFILL-FOUNDATION-F00	9431.60	4.70
EXCAVATION-FOUNDATION-F00	28791.14	14.36
FORMWORK-SLAB-F00	14595.00	7.28
FORMWORK-SLAB-F01	9478.70	4.73
FORMWORK-SLAB-F02	8355.48	4.17
POUR-CONCRETE-FOUNDATION-F00	8960.00	4.47
REINFORCING-STEEL-SLAB-F00	12605.32	6.29
REINFORCING-STEEL-SLAB-F01	14049.66	7.01
REINFORCING-STEEL-SLAB-F02	12704.85	6.34

Figure 6-43. Example of a Relative Cost Report

6.2.4.4 Explanation CONSTRUCTION PLANEX provides the user with some explanation of the results of the planning process using a set of *explanation functions* accessed through the *Explain* menu. In the current version of the system, there is no general mechanism for explanation, but a set of specific explanation functions exists for each of the attributes of a context object. Consider the explanation given for a quantity take-off computation shown in Figure 6-44 (user input is underlined). The explanation function displays the values of the *why-formula* and *why-amount* slots of the element activity schema. First, the system displays the quantity: 648.27 pounds. Then the system displays the formula used to compute this value and the KS that was used to select the quantity take-off formula. Finally, the system indicates that data for the formula was obtained from the schema titled *F02-DE-65-1-2*, which is the design element schema linked to the particular element activity.

A second example is shown in Figure 6-45. This type of explanation is displayed when the user requests information about the crew type selected for a project activity. First, the type of crew assigned to the activity is displayed. Then the explanation function displays the names of other project activity schemas that are linked to the same technology group as the project activity being considered. Finally, the system prints the KS that was used to select the crew (*crew-remove-forms-06*).

EXPLAINING AMOUNTS OF WORK OF ELEMENT ACTIVITIES

EA > F02-EA-3-210-00-65-1-2
 with name REINFORCING-STEEL-COLUMN-2 has the amount 648.27 LB

This result was obtained by evaluating formula > FORMULA-09
 which has the following syntax

((XL-DIMENSION * YL-DIMENSION * (ABS ZL-DIMENSION) * PSTEEL *
 13230 * 1.05) / 27.0)

This formula was chosen after evaluating KS > KS-Amount-3-210-0

-> Do you want to see this KS ? [y] y

```
{ { KS-Amount-3-210-0
  IS-A: KS
  KS-TYPE: FIRST
  COND-OBJECTS: CURRENT-OBJECT
  CONDITIONS: (MY-MEMBER NAME-CODE (60 65 80 81))
  LHS-RULES: (T)
  RHS-RULES: (X)
  ACTIONS: FORMULA-09 }}
```

-> Do you want to display schemas used to evaluate this KS ? [y] n

Data for the formula was inherited from
 DE > F02-DE-65-1-2

Do you want to see this DE ? [y] n

Figure 6-44. Example of Explanation for Quantity Take-Offs

6.3 Use of CONSTRUCTION PLANEX

6.3.1 Overall Behavior

This section describes how the components of the CONSTRUCTION PLANEX system are used to assist in the construction planning process. As shown in Figure 6-46, user interaction with the system is classified into three levels:

- a *strategic* level where control operators generate networks of domain operators to satisfy goals or propagate context changes;
- an *operative* level where the user executes operators or directly modifies the information stored in context objects; and

EXPLAINING TECHNOLOGY OF PROJECT ACTIVITIES

```

PA > F01-PA-30-65
with name REMOVE-FORMS-COLUMNS-F01 has the technology
> CREW-REMOVE-FORMS-06

This project activity is grouped below technology group
> GROUP-TECHNOLOGY-4
and has as brothers the following project activities

(F00-PA-30-65 F00-PA-30-80
 F01-PA-30-65 F01-PA-30-80
 F02-PA-30-65 F02-PA-30-80)

The KS used to select technology was > KS-Tech-30

Do you want to see this KS ? [y] y

{{ KS-Tech-30
  IS-A: KS
  KS-NAME: KS-Tech-30
  KS-TYPE: FIRST
  COND-OBJECTS: CURRENT-OBJECT FUNCTION NONE CURRENT-OBJECT
  CONDITIONS: (= PA-CODE 30-60)
               (= (CHECK-ELEMENTS-FOUNDATION CURRENT-OBJECT) T)
               (MY-MEMBER PA-CODE (30-65 30-80))
  LHS-RULES: (T T F) (F I F) (F I T)
  RHS-RULES: (X I) (X I) (I X)
  ACTIONS: CREW-REMOVE-FORMS-05 CREW-REMOVE-FORMS-06}}

Do you want to display schemas used to evaluate this KS ? [y] n

```

Figure 6-45. Example of Explanation for Technology Decisions

- an *interface* level where the results of the planning process are displayed or explained.

These levels are identical to the three levels of the hybrid model for process planning described in Section 3.1.4.

The strategic planning level is provided through the options of the CONTROL PANEL (CP). The user selects the "Interactive Control Panel" option of the *Operations* menu. No direct modification of context objects other than the *agenda* is permitted. The *Forward Propagation Operator* (FPO), the *Backward Search Operator* (BSO) and the *Network Interpretation Operator* (NIO) may be executed at the strategic level. Their output is stored in the *agenda* for use by the *Domain Operator Executor* (DOE) of the operative level. The strategic layer is used for two main purposes: to propagate context changes and to

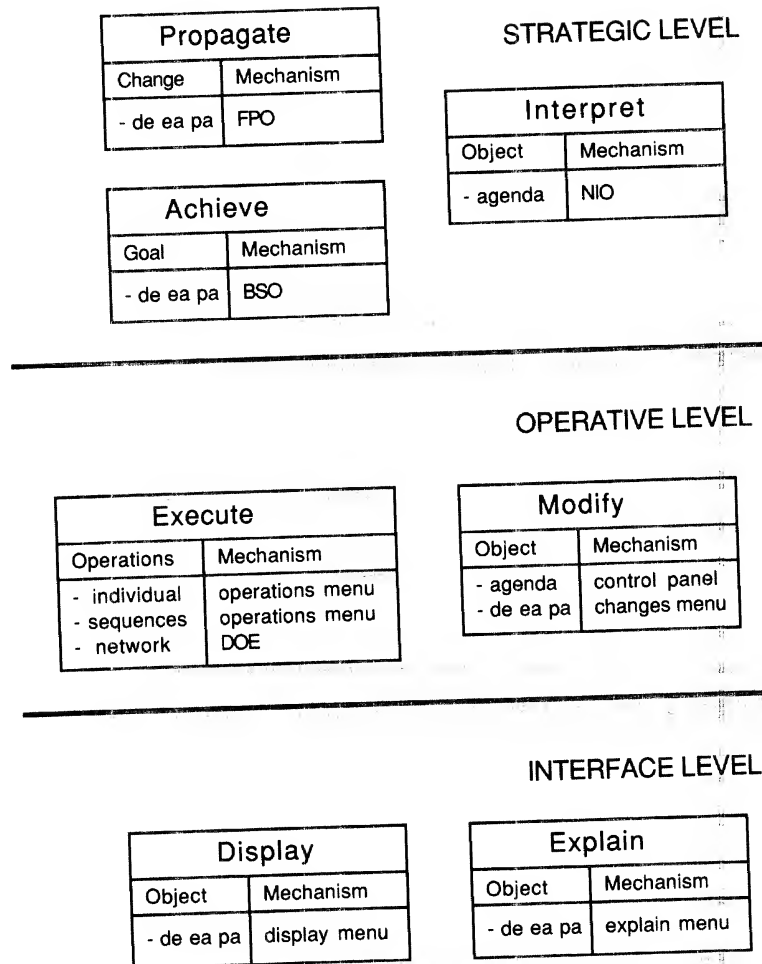


Figure 6-46. Overall Behavior of the CONSTRUCTION PLANEX System

achieve goals. For example, suppose that the planner decides to change the type of crew assigned to a particular project activity after an initial plan has been generated. CONSTRUCTION PLANEX would use the FPO and NIO to identify which operators have to be executed as a result of this change (e.g., recompute the number of crews allocated to the activity, recalculate its duration).

The operative planning level is accessed through the CP and the Menus. Interactions that create and modify the elements of a project plan take place at this level. Two types of interactions are provided:

- execute one or more operators; or
- directly modify context objects.

In both cases, context objects are modified and these changes are reflected in the final construction project plan. Operators may be invoked in three ways:

- Individual operators may be invoked using the *Individual Operations* menu of the system.
- Predefined sequences of operators may be invoked by using functions that execute the operators. An example of this type of interaction occurs when the "Forward Pass" option of the *Operations* menu is selected (see Section 6.3.2).
- The *Domain Operator Executor* (DOE) may be used to execute operators from the operator queue of the *agenda*. Operator execution order is determined from the precedences among the operators.

The interface level consists of passive interaction mechanisms that do not modify context objects or the *agenda*. These mechanisms facilitate the interpretation of the results of the planning process. In addition to the *Display* and *Explain* menus, the user may access other interaction mechanisms such as output graphical displays or the REPORT GENERATOR.

Overall control is provided via the set of menus. These let the user select one of the interaction mechanisms from the three levels of interaction. Once a planning step is initiated, it runs to completion. Only one task may be active at any time.

6.3.2 Execution of the System

CONSTRUCTION PLANEX has been used in three ways:

- as an automated planner;
- interactively as an intelligent planning assistant; and
- as a component of the INTEGRATED BUILDING DESIGN ENVIRONMENT (IBDE) [32].

Fully automated execution of the system is invoked by choosing the "Forward Pass" option of the *Operations* menu. The system applies domain operators in the following predefined sequence:

- Step 1. Create-DE-Tree.* Build the tree of design elements from the description of the structure.
- Step 2. Create-EAS.* Generate the set of element activities used to construct each design element.
- Step 3. Create-EA-Tree.* Link the element activities into a tree.

- Step 4. *Compute-Amount-EAS*. Compute the amount of work to be performed for each element activity.
- Step 5. *Determine-Unit-EAS*. Determine the unit of measure of the work quantities for each element activity.
- Step 6. *Determine-Material-EAS*. Select the material package used by each element activity.
- Step 7. *Create-PAS-for-EAS*. Synthesize project activities from element activities.
- Step 8. *Create-PA-Tree*. Link the project activities into a tree.
- Step 9. *Select-Technology-PAS*. Select the technologies used to construct each project activity.
- Step 10. *Compute-Amount-PAS*. Compute the quantity of work to be performed for each project activity.
- Step 11. *Determine-Recommended-Duration-PAS*. Generate a recommended activity duration for each project activity.
- Step 12. *Get-Duration-PAS*. Determine how many crews to allocate to each project activity.
- Step 13. *Get-Duration-EAS*. Estimate the duration of the element activities.
- Step 14. *Get-Successors-PAS*. Establish precedences among the project activities.
- Step 15. *Get-Lags-PAS*. Derive the leads and lags between consecutive project activities.
- Step 16. *Get-Cost-PAS*. Compute an estimated cost for project activities.
- Step 17. *Floyd-Warshall*. Apply the Floyd-Warshall algorithm to schedule the project.
- Step 18. *Compute-NPV*. Compute the net present value of the project.

The only user interaction occurs at the start of the process when the user specifies the set of design elements (e.g., a floor of the building) for which the plan is to be generated. This selection is made via the *Design Element* menu. This lets the user specify the portion of the building to which the planning process is applied. Assumptions regarding the details of planning decisions are used in automated planning; e.g., when determining durations for project activities in this mode, CONSTRUCTION PLANEX assumes that the user wants to avoid fractional working days. This reduces the number of questions the user is asked.

In interactive execution, the user invokes distinct operators directly from the *Individual Operations* menu. The user selects individual operators and the range of objects to which the operator will be applied, one at a time. The user is responsible for selecting operators in proper sequences to generate the project plan.

These individual operators may query the user to control details of the planning process. For example, in determining activity durations (as described in Section 6.1.3), the system estimates the recommended activity durations using *Duration* KSs and then computes the required numbers of crews. During interactive execution, the system lets the user change the values of these variables and asks whether fractional working days should be eliminated or not (as shown in Figure 6-47).

A third use of CONSTRUCTION PLANEX is as part of the INTEGRATED BUILDING DESIGN ENVIRONMENT, a vertically integrated set of knowledge-based tools for building design. Actual use of CONSTRUCTION PLANEX does not vary from that described above. The only distinction is that the description of the building is prepared directly by other components of the IBDE [32].

- ARCHPLAN [89] produces the conceptual design of a building on the basis on user requirements;
- STANLEY and STRYPES synthesize a preliminary structural design using information about the three-dimensional structural grid (proposed by ARCHPLAN) and wind and live loads;
- SPEX [38] produces a preliminary design of the individual components of the structural system; and
- FOOTER produces a preliminary design of the building foundation.

Together these systems produce the design element information needed in construction planning. To integrate CONSTRUCTION PLANEX with the other IBDE processes, two types of operators were added to the system:

- *disaggregation operators* decompose the building data produced by STANLEY and STRYPES into groups of design elements on each floor; and
- *aggregation operators* compute the cost and duration of the aggregated groups of design elements.

With the exception of these additions, CONSTRUCTION PLANEX was used as developed.

6.3.3 Examples Tested

CONSTRUCTION PLANEX has been used to plan the excavation and erection of approximately fifteen different buildings. Most tests were for concrete-frame office buildings. In one test, part of an actual building was used. Information for this example was obtained from a local contractor. In other tests, building descriptions were directly input using the INPUT GENERATOR [116] or were synthesized by the IBDE (see above).

It is difficult to compare the results of the system with aggregate estimating cost data because CONSTRUCTION PLANEX only plans for the excavation and

The system assigns to activity
POUR-CONCRETE-FOUNDATION-F00 a default duration of 5 days.
Number of crews needed to satisfy this duration are 0.89 crews of
type CREW-POUR-CONCRETE-05

***** Duration Information for
PA POUR-CONCRETE-FOUNDATION-F00

```
- Crew                CREW-POUR-CONCRETE-05
- Components of Crew  ((6 LABORERS) (1 CEMENT-FINISHER)
                      (1 OPERATOR-EQUIPMENT)
                      (2 GAS-ENGINE-VIBRATORS)
                      (1 CONCRETE-PUMP))
- Number of Crews     0.89
- Number of days      5
```

Would you like to change any of these settings ? [n] n

CONSTRUCTION PLANEX does not allow to have fractional crews in
project activities. If you want fractional crews you have to split
this activity using the SPLITTING MENU.
New number of crews is > 1.0

***** Duration Information for
PA POUR-CONCRETE-FOUNDATION-F00

```
- Crew                CREW-POUR-CONCRETE-05
- Components of Crew  ((6 LABORERS) (1 CEMENT-FINISHER)
                      (1 OPERATOR-EQUIPMENT)
                      (2 GAS-ENGINE-VIBRATORS)
                      (1 CONCRETE-PUMP))
- Number of Crews     1.0
- Number of days      4.4
```

Would you like to change any of these settings ? [n] n

**** The real duration of PA F00-PA-50-60 is 35.6 hours
**** or 4.4 days

Would you like to use overtime to eliminate day fractions? n

Figure 6-47. Example of Duration Estimation in the Interactive Mode

erection of structural building elements and its aggregate costs exclude other elements such as finishes and mechanical systems. The next section describes a test case for an eight-story steel-frame office building and shows the calculations to estimate the total cost of the building based on average unit costs [71]. It shows that the cost estimate produced by CONSTRUCTION PLANEX is within the range of answers obtained using aggregated square-foot costs [70].

6.4 Example Problem

The use of the CONSTRUCTION PLANEX system will be illustrated with the example steel-frame office building of Figure 6-48. This example was used in *Means Square Foot Costs* [70, p. 216] to illustrate the square-foot cost estimating process. A summary of the building features are:

- 8 stories;
- story height is 12'-0";
- floor dimensions are 100'-0" \times 60'-0";
- transverse framing yields 4 bays, 20'-0" on center;
- longitudinal framing yields 2 bays, 30'-0" on center;
- concrete column footings are of three sizes:
 - corner footings: 10'-6" \times 10'-6" \times 2'-1";
 - exterior footings: 12'-0" \times 12'-0" \times 3'-1"; and
 - interior footings: 13'-6" \times 13'-6" \times 3'-5";
- wide-flange steel columns (W14x120) and steel beams (W14x120) frame the structure;
- longitudinal bay floor framing is 6'-0" on centers;
- concrete floor slabs are 6" thick;
- concrete element properties are:
 - 4000 psi concrete; and
 - 1% reinforcing steel (by area).

A total of 35 design element schemas are used to describe the building (see Figure 6-49). Footings are described using three design element schemas: one for each type of footing (corner, exterior and interior). Beams are described using two design element schemas per floor: one storing descriptions of the forty-four (44) 25'-0" beams; and one storing descriptions of the ten (10) 30'-0" beams (lengths are column-line to column-line dimensions). Columns and slabs are each represented by one design element schema per floor.

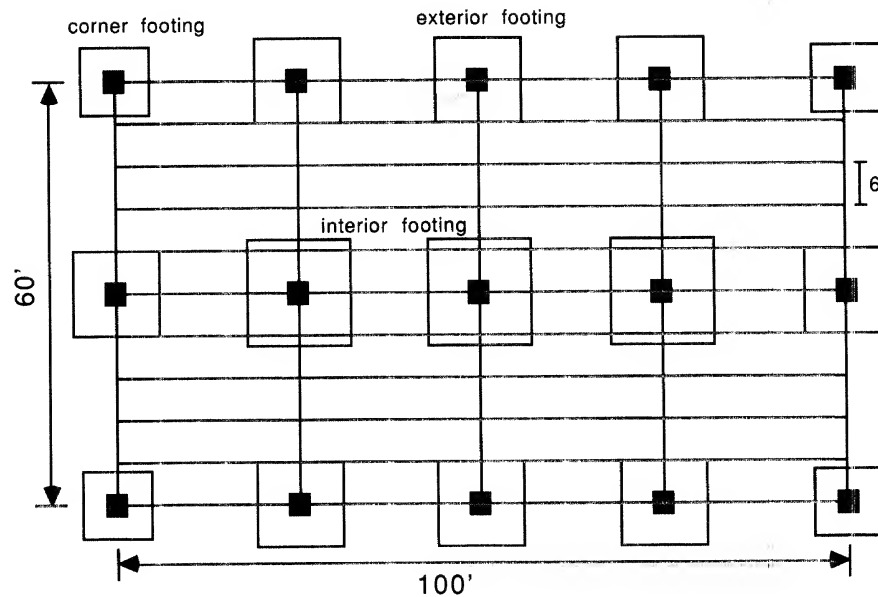


Figure 6-48. Floor Plan of the Example Building

6.4.1 Obtaining an Initial Project Plan

CONSTRUCTION PLANEX obtains an initial construction plan for the building by performing the following steps:

- Step 1. Input Building Description.* The process starts with the system inputting the definition of the 35 design element schemas described above.
- Step 2. Group Design Elements.* Design element schemas are grouped with respect to their location and type.
- Step 3. Create Element Activities.* Schemas for the 93 element activities (shown in Figure 6-50) are created on the basis of the characteristics of the individual design elements.
- Step 4. Group Element Activities.* Element activities are organized into a tree structure.
- Step 5. Compute Quantity Take-Offs.* The work quantities for the element activities are computed using the geometric descriptions of the design elements.
- Step 6. Select Material Packages.* Appropriate material packages are selected for the element activities.
- Step 7. Aggregate Element Activities.* The 93 element activity schemas are aggregated into the 63 project activity schemas shown in Figure 6-51.

Floor	Type of Design Element	Different Schemas per Floor	Total Schemas Used
F01	Column Footings	3	3
F01	Beams	2	2
F01	Columns	1	1
F01	Slabs	1	1
F02 -> F08	Beams	2	14
F02 -> F08	Columns	1	7
F02 -> F08	Slabs	1	7
Total			35

Figure 6-49. Number of Design Element Schemas Used in the Example

- Step 8. Select Crew Types.* Appropriate construction crews for the project activities are selected.
- Step 9. Aggregate Quantity Take-Offs.* The sum of the work quantities for the element activities grouped into each project activity is stored in the corresponding project activity schema.
- Step 10. Recommend Durations.* An appropriate duration for each project activity is determined on the basis of the corresponding quantity of work.
- Step 11. Determine Crew Numbers.* The number of crews allocated to each project activity is computed using its recommended duration and the adjusted productivity of the crew selected for the activity. The number of crews is stored in the project activity schema.
- Step 12. Estimate Activity Durations.* The duration of each project activity, normal working hours and overtime hours are determined. Durations for the element activities are computed.
- Step 13. Link Activities.* The 63 project activities are linked into a project activity network.
- Step 14. Get Precedences.* The types of leads and lags between each project activity and its successors (e.g., Start-to-Start) are determined.
- Step 15. Prepare Schedule.* A scheduling algorithm is applied to the network to obtain the earliest and latest start times for the project activities. The earliest completion time of the project is estimated to be 45 working days for excavation and structural erection using a fast-track schedule.
- Step 16. Compute Costs.* Material and labor costs are aggregated and the estimated total *bare* cost (i.e., overhead and profit are excluded) is computed. The estimated total cost for excavation and erection is \$983,393.

Type of Design Element	Number of Design Element Schemas	Type of Element Activity	Number of Element Activity Schemas
Column Footing	3	Excavate Column Footing	3
		Dispose-Of-Excavation	
		Column Footing	3
		Borrow Material Column Footing	3
		Place Forms Column Footing	3
		Reinforce Column Footing	3
		Pour Concrete Column Footing	3
		Remove Forms Column Footing	3
Beams	16	Erection Steel Beams	16
		Join Beams with Columns	16
Columns	8	Erection Steel Columns	8
Slabs	8	Place Forms Slabs	8
		Pour Concrete Slabs	8
		Reinforce Slabs	8
		Remove Forms Slabs	8
Total	35		93

Figure 6-50. Types of Element Activities Created for Each Type of Design Element

Type of Element Activity	Number of Element Activity Schemas	Type of Project Activity	Number of Project Activity Schemas
Excavate Column Footing	3	Excavation Foundation	1
Dispose-Of-Excavation			
Column Footing	3	Haul Excavation Foundation	1
Borrow Material			
Column Footing	3	Backfill Foundation	1
Place Forms Column Footing	3	Formwork Foundation	1
Reinforce Column Footing	3	Reinforcing Steel Foundation	1
Pour Concrete Column Footing	3	Pour Concrete Foundation	1
Remove Forms Column Footing	3	Remove Forms Foundation	1
Erection Steel Beams	16	Erection Steel Beams	8
Join Beams with Columns	16	Join Steel	8
Erection Steel Columns	8	Erection Steel Columns	
		Diagonals	8
Place Forms Slabs	8	Formwork Slabs	8
Reinforce Slabs	8	Reinforce Slabs	8
Pour Concrete Slabs	8	Pour Concrete Slabs	8
Remove Forms Slabs	8	Remove Forms Slabs	8
Total	93		63

Figure 6-51. Types of Project Activities Created for Each Type of Element Activity

Step 17. Compute Net Present Value. The Net Present Value (NPV) of the project is computed using the scheduling and cost information from the project activity schemas. The NPV is estimated to be \$977,914.¹⁰ This value is less than the total cost (\$983,393) because the total cost ignores the time value of money.

Step 18. Output Results. A set of displays and reports illustrating the results of the planning process are produced by the GANTT, REPORT GENERATOR, ANIMATOR and the passive output graphical processors.

The schedule results of the planning process are shown in Figure 6-52¹⁰. The report was prepared by the REPORT GENERATOR. It contains seven columns:

1. "CODE" is the project activity code;
2. "LOCATION" identifies the work location (floor) of a project activity;
3. "DURATION" is the estimated project activity duration;
4. "EST" is the earliest-start-time of a project activity;
5. "EFT" is the earliest-finish-time of a project activity;
6. "LST" is the latest-start-time of a project activity; and
7. "LFT" is the latest-finish-time of a project activity.

Figure 6-53 shows a part of the cost report for the example building. The columns of this report are:

1. "NAME" identifies a project activity;
2. "COST" is the total cost of a project activity (i.e., the sum of the crew and material costs); and
3. "PERCENT" is the percent cost of a project activity.

Figure 6-54 shows a portion of a report describing crews. It contains six columns:

1. "CODE" is the project activity code;
2. "LOCATION" identifies the work location (floor) of a project activity;
3. "AMOUNT" is the work quantity for a project activity;
4. "CREW" designates the name of the crew used to perform a project activity;
5. "NO." is the number of crews assigned to a project activity; and
6. "DUR." is the estimated duration (in days) of a project activity.

The report shows that the same crew type is selected for all the activities having the same activity code. However, different crews are assigned to activities of

¹⁰ All of the figures illustrating output reports have been edited for publication. Typically, these partial reports only contain information for the lower floors of the building.

CODE	LOCATION	DURATION	EST	EFT	LST	LFT
10-60	F01	4.4	0.00	4.43	0.00	4.43
20-60	F01	5.0	0.17	5.21	0.17	5.21
40-60	F01	5.9	0.38	6.30	0.38	6.30
15-60	F01	2.3	2.43	4.69	47.37	49.63
55-65	F01	0.2	6.07	6.32	6.07	6.32
50-60	F01	2.2	6.08	8.32	46.95	49.18
55-81	F01	1.9	6.10	8.05	6.10	8.05
20-80	F01	5.3	6.14	11.45	18.35	23.66
58-81	F01	2.0	6.38	8.38	22.01	24.01
30-60	F01	1.7	7.08	8.76	47.95	49.63
55-65	F02	0.2	7.81	8.06	7.81	8.06
55-81	F02	1.9	7.85	9.79	7.85	9.79
20-80	F02	5.3	7.88	13.19	18.35	23.66
58-81	F02	2.0	8.12	10.12	22.01	24.01
17-60	F01	0.4	8.76	9.15	49.63	50.01
40-80	F01	2.8	8.99	11.80	21.20	24.01
55-65	F03	0.2	9.56	9.81	9.56	9.81
55-81	F03	1.9	9.59	11.53	9.59	11.53
20-80	F03	5.3	9.62	14.93	18.35	23.66
58-81	F03	2.0	9.87	11.87	22.01	24.01
40-80	F02	2.8	10.74	13.54	21.20	24.01
55-65	F04	0.2	11.30	11.55	11.30	11.55
55-81	F04	1.9	11.33	13.28	11.33	13.28
20-80	F04	5.3	11.37	16.68	18.35	23.66
58-81	F04	2.0	11.61	13.61	22.01	24.01
50-80	F01	0.5	11.80	12.33	24.01	24.54
40-80	F03	2.8	12.48	15.29	21.20	24.01
55-65	F05	0.2	13.05	13.30	13.05	13.30
55-81	F05	1.9	13.08	15.02	13.08	15.02
20-80	F05	5.3	13.11	18.42	18.35	23.66
58-81	F05	2.0	13.36	15.36	22.01	24.01
50-80	F02	0.5	13.54	14.07	24.01	24.54
40-80	F04	2.8	14.23	17.03	21.20	24.01
50-80	F03	0.5	15.29	15.81	24.01	24.54
40-80	F05	2.8	15.97	18.77	21.20	24.01
50-80	F04	0.5	17.03	17.56	24.01	24.54
50-80	F05	0.5	18.77	19.30	24.01	24.54
30-80	F01	5.0	32.80	37.80	45.01	50.01
30-80	F02	5.0	34.54	39.54	45.01	50.01
30-80	F03	5.0	36.29	41.29	45.01	50.01
30-80	F04	5.0	38.03	43.03	45.01	50.01
30-80	F05	5.0	39.77	44.77	45.01	50.01

Figure 6-52. Partial Scheduling Report for the Example Building

NAME	COST	PERCENT
BACKFILL-FOUNDATION-F01	799.28	0.08
ERECTION-STEEL-BEAMS-F01	60281.00	6.13
ERECTION-STEEL-BEAMS-F02	60281.00	6.13
ERECTION-STEEL-BEAMS-F03	60281.00	6.13
ERECTION-STEEL-BEAMS-F04	60281.00	6.13
ERECTION-STEEL-BEAMS-F05	60281.00	6.13
ERECTION-STEEL-COLUMNS-DIAGONALS-F01	7750.80	0.79
ERECTION-STEEL-COLUMNS-DIAGONALS-F02	7750.80	0.79
ERECTION-STEEL-COLUMNS-DIAGONALS-F03	7750.80	0.79
ERECTION-STEEL-COLUMNS-DIAGONALS-F04	7750.80	0.79
ERECTION-STEEL-COLUMNS-DIAGONALS-F05	7750.80	0.79
EXCAVATION-FOUNDATION-F01	2984.76	0.30
FORMWORK-FOUNDATION-F01	8303.08	0.84
FORMWORK-SLABS-F01	25139.89	2.56
FORMWORK-SLABS-F02	25139.89	2.56
FORMWORK-SLABS-F03	25139.89	2.56
FORMWORK-SLABS-F04	25139.89	2.56
FORMWORK-SLABS-F05	25139.89	2.56
HAUL-EXCAVATION-FOUNDATION-F01	2976.88	0.30
POUR-CONCRETE-FOUNDATION-F01	16413.21	1.67
POUR-CONCRETE-SLABS-F01	7772.23	0.79
POUR-CONCRETE-SLABS-F02	7772.23	0.79
POUR-CONCRETE-SLABS-F03	7772.23	0.79
POUR-CONCRETE-SLABS-F04	7772.23	0.79
POUR-CONCRETE-SLABS-F05	7772.23	0.79
REINFORCING-STEEL-FOUNDATION-F01	11675.63	1.19
REINFORCING-STEEL-SLABS-F01	5528.82	0.56
REINFORCING-STEEL-SLABS-F02	5528.82	0.56
REINFORCING-STEEL-SLABS-F03	5528.82	0.56
REINFORCING-STEEL-SLABS-F04	5528.82	0.56
REINFORCING-STEEL-SLABS-F05	5528.82	0.56
REMOVE-FORMS-FOUNDATION-F01	1335.17	0.14
REMOVE-FORMS-SLABS-F01	3840.00	0.39
REMOVE-FORMS-SLABS-F02	3840.00	0.39
REMOVE-FORMS-SLABS-F03	3840.00	0.39
REMOVE-FORMS-SLABS-F04	3840.00	0.39
REMOVE-FORMS-SLABS-F05	3840.00	0.39
JOIN-STEEL-F01	7047.36	0.72
JOIN-STEEL-F02	7047.36	0.72
JOIN-STEEL-F03	7047.36	0.72
JOIN-STEEL-F04	7047.36	0.72
JOIN-STEEL-F05	7047.36	0.72

Figure 6-53. Partial Cost Report for the Example Building

CODE	LOCATION	AMOUNT	CREW	NO.	DUR.
17-60	F01	229.02	CREW-BACKFILL-FOUNDATION-05	1.0	0.4
55-81	F01	1400.00	CREW-ERECT-BEAM-COLUMN-02	1.0	1.9
55-81	F02	1400.00	CREW-ERECT-BEAM-COLUMN-02	1.0	1.9
55-81	F03	1400.00	CREW-ERECT-BEAM-COLUMN-02	1.0	1.9
55-81	F04	1400.00	CREW-ERECT-BEAM-COLUMN-02	1.0	1.9
55-81	F05	1400.00	CREW-ERECT-BEAM-COLUMN-02	1.0	1.9
55-65	F01	180.00	CREW-ERECT-BEAM-COLUMN-02	1.0	0.2
55-65	F02	180.00	CREW-ERECT-BEAM-COLUMN-02	1.0	0.2
55-65	F03	180.00	CREW-ERECT-BEAM-COLUMN-02	1.0	0.2
55-65	F04	180.00	CREW-ERECT-BEAM-COLUMN-02	1.0	0.2
55-65	F05	180.00	CREW-ERECT-BEAM-COLUMN-02	1.0	0.2
10-60	F01	442.84	CREW-EXCAVATION-FOUNDATION-05	1.0	4.4
20-60	F01	2086.20	CREW-FORMWORK-07	1.0	5.0
20-80	F01	6000.00	CREW-FORMWORK-06	2.0	5.3
20-80	F02	6000.00	CREW-FORMWORK-06	2.0	5.3
20-80	F03	6000.00	CREW-FORMWORK-06	2.0	5.3
20-80	F04	6000.00	CREW-FORMWORK-06	2.0	5.3
20-80	F05	6000.00	CREW-FORMWORK-06	2.0	5.3
15-60	F01	487.13	CREW-HAUL-FOUNDATION-03	3.0	2.3
50-60	F01	234.64	CREW-POUR-CONCRETE-06	1.0	2.2
50-80	F01	111.11	CREW-POUR-CONCRETE-06	2.0	0.5
50-80	F02	111.11	CREW-POUR-CONCRETE-06	2.0	0.5
50-80	F03	111.11	CREW-POUR-CONCRETE-06	2.0	0.5
50-80	F04	111.11	CREW-POUR-CONCRETE-06	2.0	0.5
50-80	F05	111.11	CREW-POUR-CONCRETE-06	2.0	0.5
40-60	F01	32595.25	CREW-RE-STEEL-05	1.0	5.9
40-80	F01	15435.00	CREW-RE-STEEL-05	1.0	2.8
40-80	F02	15435.00	CREW-RE-STEEL-05	1.0	2.8
40-80	F03	15435.00	CREW-RE-STEEL-05	1.0	2.8
40-80	F04	15435.00	CREW-RE-STEEL-05	1.0	2.8
40-80	F05	15435.00	CREW-RE-STEEL-05	1.0	2.8
30-60	F01	2086.20	CREW-REMOVE-FORMS-05	3.0	1.7
30-80	F01	6000.00	CREW-REMOVE-FORMS-06	3.0	5.0
30-80	F02	6000.00	CREW-REMOVE-FORMS-06	3.0	5.0
30-80	F03	6000.00	CREW-REMOVE-FORMS-06	3.0	5.0
30-80	F04	6000.00	CREW-REMOVE-FORMS-06	3.0	5.0
30-80	F05	6000.00	CREW-REMOVE-FORMS-06	3.0	5.0
58-81	F01	864.00	CREW-JOIN-STEEL-01	4.0	2.0
58-81	F02	864.00	CREW-JOIN-STEEL-01	4.0	2.0
58-81	F03	864.00	CREW-JOIN-STEEL-01	4.0	2.0
58-81	F04	864.00	CREW-JOIN-STEEL-01	4.0	2.0
58-81	F05	864.00	CREW-JOIN-STEEL-01	4.0	2.0

Figure 6-54. Partial Crew Report for the Example Building

the same type but associated with different design elements (e.g., foundation formwork [code "20-80"] is performed with crew "crew-formwork-07" while slab formwork [code "20-60"] uses crew "crew-formwork-06").

6.4.2 Analysis of the Results

The manner in which CONSTRUCTION PLANEX computes activity durations and costs is described below, followed by a comparison of its results with aggregate square-foot cost estimates.

The basic data used to compute activity durations and costs are stored in the crew and material package schemas. The values of productivities and costs were taken from Means's *Building Construction Cost Data* [71]. For example, consider the crew schema displayed in Figure 6-55. This crew schema is equivalent to Means's "Crew-E-2". The standard productivity of this crew is 720 lineal feet of steel erection per day. This rate corresponds to the value reported [71, p. 120] for erecting W14x120 structural members (MASTERFORMAT code "05 1250 2500"). The normal installation cost per lineal foot is equal to the labor and equipment cost of the crew (\$39.32 per man-hour) times the daily man-hours of the crew (7 workers in the crew \times 8 hours per day = 56 man-hours per day), divided by the standard productivity of 720 lineal feet per day:

$$\$39.32 \times 56 / 720 = \$3.058$$

Overtime unit costs are assumed to be 50% higher than the normal unit costs. CONSTRUCTION PLANEX selected this crew for the erection of structural members on a floor.

Consider the activity of erecting steel beams for the first floor. The system computes the amount of work for this activity by adding the quantity take-offs computed for the two steel erection element activities: one storing the amount of work for the forty-four (44) 25'-0" beams and the other storing the amount of work for the ten (10) 30'-0" beams. The total work quantity for this project activity is sum of the lengths of the beams:

$$(44 \times 25) + (10 \times 30) = 1400'$$

This quantity is used to compute the duration of the structural erection activity by dividing the quantity by the daily productivity:

$$1400 / 720 = 1.944 \text{ days}$$

where 720 lineal feet per day is the standard productivity of the crew (from Figure 6-55). The material cost of the activity is computed as quantity times cost:

$$1400 \times 40 = \$56,000$$

where \$40 per foot is the estimated local material cost of a W14x120 steel beam. Similarly, the crew cost is computed as quantity times installation cost per unit quantity:

$$(1400 \times \$3.058) = \$4,281$$

```

(defschema crew-erect-beam-column-02
  (is-a      crew)
  (component-names (1 struct-steel-foreman)
                   (4 struct-steel-workers)
                   (1 equip-oper-crane)
                   (1 equip-oper-oiler)
                   (1 crane-90-tons))
  (std-productivity 720)
  (prod-unit      ft/day)
  (normal-cost     3.058)
  (overtime-cost   4.587))

```

Figure 6-55. Crew for Erecting Structural Steel Members

where \$3.058 is the normal crew cost per foot of beam. Finally, the total cost of the activity is computed as the sum of the material and labor costs:

$$(\$4,281 + \$56,000) = \$60,281$$

This corresponds to the cost for "ERECTION-STEEL-BEAMS-F01" in Figure 6-53.

The example building was used in *Means Square Foot Costs* to illustrate the square-foot cost estimating process [70, pp. 216-224]. In the example, cost of the structural elements is 23% of the total bare cost of the project. Using the NPV cost of the structural elements, the estimated total cost of the building is computed as:

$$\$979,914 / 0.23 = \$4,260,496$$

The building floor area is:

$$60 \times 100 \times 8 = 48,000 \text{ square feet}$$

Thus, the cost per square foot is:

$$4,260,496 / 48,000 = \$88.76$$

This value is higher than the \$57 per square-foot average cost for 5- to 10-story buildings, but falls within the cost ranges (\$32-\$105 per square foot) reported in the literature [70, p. 164].

6.4.3 Modifications to the Project Plan

Modifying the initial plan produced by CONSTRUCTION PLANEX is illustrated with a simple example. Suppose that the user wants to investigate the effect of using a different crew for structural erection. This situation might arise if the most appropriate equipment is not available. This alternate crew is designated "crew-erect-beam-column-01" and has a standard productivity of 500 lineal feet per day (less than the 720 lineal feet per day used in formulating the initial plan). Assume that the corresponding crew schema is already defined in the context. The process of modifying the initial plan proceeds as follows:

Step 1. The user selects the *Change* menu from the *Top* menu and indicates that he wants to change the technology of a project activity.

- Step 2.* A menu with all project activities is displayed and the user selects one of the steel erection activities (code "55").
- Step 3.* The user inputs the new crew type designation for the activity. This value is stored in the *group-technology* object linked to the selected project activity. This single substitution results in the *effect* list "(pa-schema technology filled)" being inserted in the *context-changes* slot of *agenda*.
- Step 4.* The user executes the *Forward Propagation Operator* (FPO) to obtain a network of global changes and operators. This control operator inserts 56 (*operator object*) pairs in the *agenda*. Figure 6-56 shows some of these operator pairs. In order to maintain consistency in the context, CONSTRUCTION PLANEX must recalculate the durations and costs of the 16 project activities that used the crew "crew-erect-beam-column-02" (1 per beam and 1 per column for the 8 floors), and recompute the durations of the 24 element activities linked to these 16 project activities (longitudinal beams, transverse beams and columns for each floor).
- Step 5.* The user executes the *Network Interpretation Operator* (NIO) which yields 40 precedences among the 56 operator pairs: 16 precedences between the operators that compute the durations of project activities and those that compute their associated costs; and 24 precedences between the operators that compute the duration of project activities and those that compute the duration of element activities. Some of these precedences are shown in Figure 6-57.
- Step 6.* The user executes the *Domain Operator Executor* (DOE) to propagate the effects of the crew technology change.

By executing these six steps, the user has introduced the desired change and used the control operators of PLANEX to propagate the effects of the change and maintain the consistency in the context. The only operators that remain to be executed are the *Floyd-Warshall* scheduling algorithm and the *Compute-NPV* operator (since they do not have *Domain Operator Schemas*, nothing automatically triggers their execution). Results of executing these operators are shown in Figures 6-58 and 6-59. In terms of schedule and cost, the completion time of the project increases by five days (from 50.01 days to 55.12 days), and the NPV of the project increases by 1.6% (from \$977,914 to \$993,862). These increases in time and cost are due to the extra hours of work in steel erection.

```

;
; Recompute Cost of Project Activities
;
(GET-COST-PAS F05-PA-55-65)
(GET-COST-PAS F05-PA-55-81)
(GET-COST-PAS F04-PA-55-65)
(GET-COST-PAS F04-PA-55-81)
(GET-COST-PAS F03-PA-55-65)
(GET-COST-PAS F03-PA-55-81)
(GET-COST-PAS F02-PA-55-65)
(GET-COST-PAS F02-PA-55-81)
(GET-COST-PAS F01-PA-55-65)
(GET-COST-PAS F01-PA-55-81)
;
; Recompute Duration of Project Activities
;
(GET-DURATION-PAS F05-PA-55-65)
(GET-DURATION-PAS F05-PA-55-81)
(GET-DURATION-PAS F04-PA-55-65)
(GET-DURATION-PAS F04-PA-55-81)
(GET-DURATION-PAS F03-PA-55-65)
(GET-DURATION-PAS F03-PA-55-81)
(GET-DURATION-PAS F02-PA-55-65)
(GET-DURATION-PAS F02-PA-55-81)
(GET-DURATION-PAS F01-PA-55-65)
(GET-DURATION-PAS F01-PA-55-81)
;
; Recompute Duration of Element Activities
;
(GET-DURATION-EAS F05-EA-5-120-10-65-2-1)
(GET-DURATION-EAS F05-EA-5-120-20-81-2-2)
(GET-DURATION-EAS F05-EA-5-120-20-81-2-1)
(GET-DURATION-EAS F04-EA-5-120-10-65-2-1)
(GET-DURATION-EAS F04-EA-5-120-20-81-2-2)
(GET-DURATION-EAS F04-EA-5-120-20-81-2-1)
(GET-DURATION-EAS F03-EA-5-120-10-65-2-1)
(GET-DURATION-EAS F03-EA-5-120-20-81-2-2)
(GET-DURATION-EAS F03-EA-5-120-20-81-2-1)
(GET-DURATION-EAS F02-EA-5-120-10-65-2-1)
(GET-DURATION-EAS F02-EA-5-120-20-81-2-2)
(GET-DURATION-EAS F02-EA-5-120-20-81-2-1)
(GET-DURATION-EAS F01-EA-5-120-10-65-2-1)
(GET-DURATION-EAS F01-EA-5-120-20-81-2-2)
(GET-DURATION-EAS F01-EA-5-120-20-81-2-1)

```

Figure 6-56. (Operator Object) Pairs Created by the
Forward Propagation Operator

FROM-OPERATOR	TO-OPERATOR
(GET-DURATION-PAS F04-PA-55-65)	(GET-COST-PAS F04-PA-55-65)
(GET-DURATION-PAS F04-PA-55-81)	(GET-COST-PAS F04-PA-55-81)
(GET-DURATION-PAS F03-PA-55-65)	(GET-COST-PAS F03-PA-55-65)
(GET-DURATION-PAS F03-PA-55-81)	(GET-COST-PAS F03-PA-55-81)
(GET-DURATION-PAS F02-PA-55-65)	(GET-COST-PAS F02-PA-55-65)
(GET-DURATION-PAS F02-PA-55-81)	(GET-COST-PAS F02-PA-55-81)
(GET-DURATION-PAS F01-PA-55-65)	(GET-COST-PAS F01-PA-55-65)
(GET-DURATION-PAS F01-PA-55-81)	(GET-COST-PAS F01-PA-55-81)
(GET-DURATION-PAS F04-PA-55-65)	(GET-DURATION-EAS F04-EA-5-120-10-65-2-1)
(GET-DURATION-PAS F04-PA-55-81)	(GET-DURATION-EAS F04-EA-5-120-20-81-2-2)
(GET-DURATION-PAS F04-PA-55-81)	(GET-DURATION-EAS F04-EA-5-120-20-81-2-1)
(GET-DURATION-PAS F03-PA-55-65)	(GET-DURATION-EAS F03-EA-5-120-10-65-2-1)
(GET-DURATION-PAS F03-PA-55-81)	(GET-DURATION-EAS F03-EA-5-120-20-81-2-2)
(GET-DURATION-PAS F03-PA-55-81)	(GET-DURATION-EAS F03-EA-5-120-20-81-2-1)
(GET-DURATION-PAS F02-PA-55-65)	(GET-DURATION-EAS F02-EA-5-120-10-65-2-1)
(GET-DURATION-PAS F02-PA-55-81)	(GET-DURATION-EAS F02-EA-5-120-20-81-2-2)
(GET-DURATION-PAS F02-PA-55-81)	(GET-DURATION-EAS F02-EA-5-120-20-81-2-1)
(GET-DURATION-PAS F01-PA-55-65)	(GET-DURATION-EAS F01-EA-5-120-10-65-2-1)
(GET-DURATION-PAS F01-PA-55-81)	(GET-DURATION-EAS F01-EA-5-120-20-81-2-2)
(GET-DURATION-PAS F01-PA-55-81)	(GET-DURATION-EAS F01-EA-5-120-20-81-2-1)

Figure 6-57. Partial List of Precedences Created by the
Network Interpretation Operator

6.5 Conclusions

CONSTRUCTION PLANEX demonstrates the applicability of knowledge-based expert system techniques to the problem of construction project planning. An expert system aid for this type of planning could yield substantial benefits over traditional methods by producing more consistent and detailed plans at lower costs. CONSTRUCTION PLANEX provides an operational prototype that could be extended to a full system. In addition to the CONSTRUCTION PLANEX system architecture itself, the models used in the system represent formalisms of the planning process, including: (1) the bottom-up or disaggregate activity formula-

CODE	LOCATION	DURATION	EST	EFT	LST	LFT
10-60	F01	4.4	0.00	4.43	0.00	4.43
20-60	F01	5.0	0.17	5.21	0.17	5.21
40-60	F01	5.9	0.38	6.30	0.38	6.30
15-60	F01	2.3	2.43	4.69	52.48	54.73
55-65	F01	0.4	5.96	6.32	5.96	6.32
50-60	F01	2.2	5.97	8.21	52.05	54.29
55-81	F01	2.8	5.99	8.79	5.99	8.79
20-80	F01	5.3	6.03	11.34	23.46	28.77
30-60	F01	1.7	6.97	8.65	53.05	54.73
58-81	F01	2.0	7.12	9.12	27.12	29.12
55-65	F02	0.4	8.45	8.81	8.45	8.81
55-81	F02	2.8	8.48	11.28	8.48	11.28
20-80	F02	5.3	8.52	13.83	23.46	28.77
17-60	F01	0.4	8.65	9.04	54.73	55.12
40-80	F01	2.8	8.88	11.69	26.31	29.12
58-81	F02	2.0	9.61	11.61	27.12	29.12
55-65	F03	0.4	10.94	11.30	10.94	11.30
55-81	F03	2.8	10.97	13.77	10.97	13.77
20-80	F03	5.3	11.01	16.32	23.46	28.77
40-80	F02	2.8	11.37	14.18	26.31	29.12
50-80	F01	0.5	11.69	12.22	29.12	29.65
58-81	F03	2.0	12.10	14.10	27.12	29.12
55-65	F04	0.4	13.43	13.79	13.43	13.79
55-81	F04	2.8	13.46	16.26	13.46	16.26
20-80	F04	5.3	13.50	18.81	23.46	28.77
40-80	F03	2.8	13.86	16.67	26.31	29.12
50-80	F02	0.5	14.18	14.71	29.12	29.65
58-81	F04	2.0	14.59	16.59	27.12	29.12
40-80	F04	2.8	16.35	19.16	26.31	29.12
50-80	F03	0.5	16.67	17.20	29.12	29.65
50-80	F04	0.5	19.16	19.69	29.12	29.65
30-80	F01	5.0	32.69	37.69	50.12	55.12
30-80	F02	5.0	35.18	40.18	50.12	55.12
30-80	F03	5.0	37.67	42.67	50.12	55.12
30-80	F04	5.0	40.16	45.16	50.12	55.12

Figure 6-58. Partial Scheduling Report for the Revised Project Plan

*** Financial Results *****

With

Annual MARR > 0.10

Inflation > 0.04

NPV Costs is \$993,862 (not including overhead and profit)

Figure 6-59. Output of the *Compute-NPV* Operator for the Revised Project Plan

tion model; (2) algorithms for strategic nonlinear planning; and (3) the unified activity network model. For commercial applications, the knowledge base in CONSTRUCTION PLANEX would have to be substantially expanded; the current version only plans excavation and structural erection. Extending the system to handle project monitoring and control is also possible.

The most fundamental contribution of this work is CONSTRUCTION PLANEX itself: a conceptual design of an integrated system for construction project planning and a realization of that design in a prototype system that can plan the excavation and structural erection of concrete and steel-frame buildings. When compared to other tools for construction project planning, CONSTRUCTION PLANEX has several advantages:

- CONSTRUCTION PLANEX is the first system that integrates all the elements of the construction planning process into a unified modeling and planning system. To generate a process plan, the system formulates activity networks, selects technologies and construction methods, estimates activity durations and costs, and prepares project schedules. Other construction planning systems are applicable to only parts of the whole process. In particular, most commercial systems are limited to performing scheduling computations and require the user to formulate the project plan and provide it as input to the program.
- While the scope of the prototype implementation is limited, development of CONSTRUCTION PLANEX required that the types of knowledge, problem-solving operators and representational structures be identified and formalized. The resulting formal model and structure of the construction planning problem are more general than those used in the prototype. This model can be used to extend CONSTRUCTION PLANEX, or as the basis for developing similar tools in other construction planning domains.

In terms of existing tools for construction project planning, CONSTRUCTION PLANEX includes an important feature. It provides a framework which incorporates both the *time value* of money and *value engineering* in planning. Current tools either do not provide a mechanism to consider these effects or do not provide the flexibility of CONSTRUCTION PLANEX.

As CONSTRUCTION PLANEX is only a prototype, there are several extensions which are desirable, related to both its depth and problem-solving capabilities:

- The knowledge in the system is limited to excavation and structural erection. Expansion of the knowledge base to include other aspects of building construction requires that additional design elements and activities be defined.
- The knowledge in the current prototype was provided by an experienced planner. However, it has never been verified, has not been refined by incorporating results from previous projects, and it is idiosyncratic. Verification and refinement of the knowledge base would improve the performance of the system.

- CONSTRUCTION PLANEX considers the type and location of activities during planning, but does not reason about the geometry of the elements. More comprehensive geometric reasoning could be useful in some planning applications.
 - In CONSTRUCTION PLANEX, resource leveling and resource allocation are still done by hand (albeit with the interactive GANTT scheduler). Inclusion of resource allocation methods and resource leveling algorithms would further enhance the capabilities of CONSTRUCTION PLANEX.
 - CONSTRUCTION PLANEX does not use a formal database management system (DBMS) to store planning data such as unit costs. Indeed, a DBMS could be used to store and retrieve objects such as knowledge sources.
-

11

7 HARNESS PLANEX: An Expert System for Electrical Wire Harness Process Planning

An example application of PLANEX for planning the manufacture of products is HARNESS PLANEX, an expert system that generates activity plans for manufacturing automotive electrical harnesses. Harnesses are used in automobiles to transmit electrical current. A simplified example of a harness is shown in Figure 7-1. A typical harness is composed of wires, terminals, connectors, molding, splices, tubing and tape, but may contain other electrical components such as light bulbs or diodes. Harness manufacturers are typically subcontractors to one or more vehicle manufacturers.

When a harness manufacturer receives the drawings and specifications of a harness, he must plan how to manufacture the product and estimate the cost of the harness. Typically, this planning is done by experts in the engineering department who generate a *process sheet* which describes how the harness will be manufactured. The process sheet is important to the manufacturer because it provides information about individual machine usage and material requirements. Each year, a harness manufacturer will produce hundreds of process sheets in a short period of time at the beginning of the new model production year. Process sheets must be generated as quickly as possible to produce prototype harnesses that are sent to the automobile manufacturer for final approval.

Similar to the CONSTRUCTION PLANEX system described in Chapter 6, HARNESS PLANEX receives as input information extracted from the design drawings of a particular harness and identifies the activities required for its manufac-

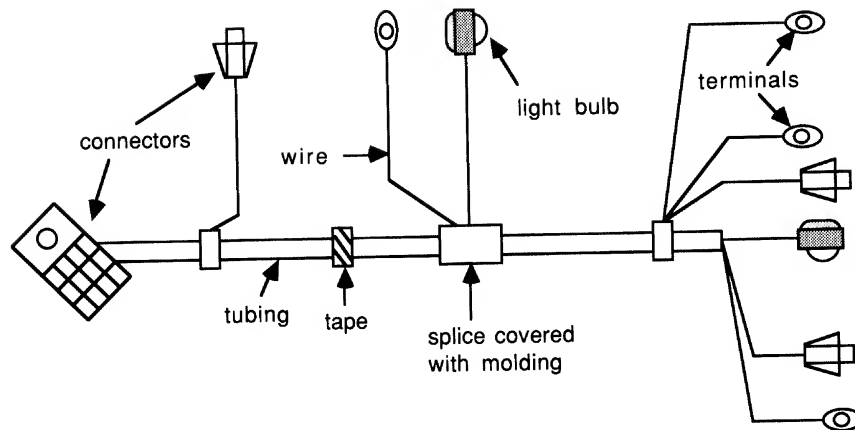


Figure 7-1. Example of an Electrical Automotive Harness

ture, selects appropriate technologies for these activities and estimates their duration and the total usage of resources. In addition, the system aggregates harness components into *subassemblies* which represent parts of the harness that may be manufactured independently before being assembled into the final harness.

The intent in the development of HARNESS PLANEX was to test the applicability of PLANEX to the automated generation of process plans. Knowledge was provided by an experienced Mexican harness manufacturer that sells harnesses in the United States and Latin America.

7.1 Models for the Harness Manufacturing Planning Process

This section discusses the models used in HARNESS PLANEX in the various stages of the planning process. Some of these models are similar to the corresponding models used in the CONSTRUCTION PLANEX system described in Section 6.1. There are some important differences, however:

- At the beginning of the production year, the harness planner is more concerned with *what* is to be done than *when* it will be done, while the construction planner is concerned with both activities and schedule throughout the planning process. The output of the harness manufacturing planning process is a set of reports identifying manufacturing activities and machine usage but without schedules for the activities.
- The set of harness manufacturing activities is small; therefore there is no need to explicitly identify activity precedences.

Because of these differences, HARNESS PLANEX uses models for only three of the four planning stages described in Section 2.3:

- *Definition of Work Tasks* models to identify manufacturing activities for harness components;
- *Choice of Technologies and Manufacturing Methods* models for the allocation of resources (machines, labor and materials) to the manufacturing activities; and
- *Estimation of Activity Durations and Costs* models to estimate expected activity durations and costs.

The modularity of the PLANEX architecture permits extensions of the system to other planning operations. The user would have to define the procedural code and the *Domain Operator Schemas* of the new operators as well as corresponding knowledge sources. For example, the operators for production line scheduling could use the data produced by the current version of HARNESS PLANEX as a basis for the job shop scheduling process. Other information such as lane layout, production volume and promised delivery dates would have to be included in the analysis.

7.1.1 Definition of Work Tasks

Generating the set of manufacturing activities for a harness begins with specifications of: (1) the individual components of the harness (e.g., wires, terminals); and (2) the harness topology. This design description is the input to a process which decomposes the harness into unitary components (see Figure 7-2). The process takes the descriptions of the characteristics of the wires in the harness (e.g., their length, gauge, type of insulation, location) and stores the information in *wire* objects. This information is used to identify some of the manufacturing activities (e.g., cutting each wire from the spool) but is insufficient to generate a complete set of manufacturing activities. Some activities such as tinning wire ends are performed on multiple wire ends simultaneously and are viewed as a single activity. Therefore, additional objects representing the relationships among the individual wires must be created and analyzed to obtain the set of work tasks.

To establish these relationships, HARNESS PLANEX decomposes each wire into three parts: its left *extreme* (i.e., end), its *body* and its right extreme. Wires are connected to each other by common terminals. To distinguish among terminals of the same type, each terminal is associated with a specific location in the harness. The resulting *terminal-location* objects represent wire connections. The set of *design elements* from which element activities are synthesized are the *body* and the *terminal-location* objects.

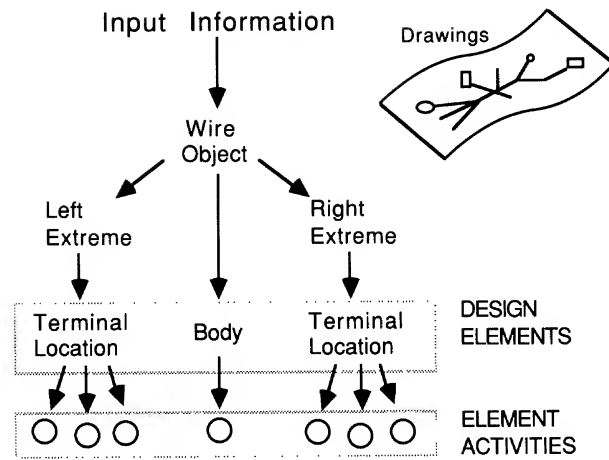


Figure 7-2. Work Task Decomposition Model of HARNESS PLANEX

Having generated a set of design element objects describing the wires and their relationships, HARNESS PLANEX completes the definition of the work tasks by defining two types of activities:

- *body activities* associated with the manufacture of the *body* objects; and
- *extreme activities* associated with the manufacture of the *terminal-location* objects.

Once generated, these activities are used to prepare the process sheet. No activity aggregations (similar to the aggregation of *element activities* into a *project activity* in CONSTRUCTION PLANEX) are required.

There are some limitations in the work task decomposition model used by HARNESS PLANEX:

- the model does not generate activities for handling completed subassemblies; and
- the model produces only one level of activity aggregation.

The first limitation is a consequence of the characteristics of the harness manufacturing process. Generally the activities applied to completed subassemblies are performed in the assembly portion of the production line. These activities depend on both the characteristics of the harness and the type of tools used (e.g., custom jigs to hold subassemblies). A detailed description of the tools used in the assembly process is needed if the assembly activities are to be generated. This is beyond the scope of the prototype. With respect to the second limitation, there is no need to aggregate the manufacturing activities of a single harness because each activity requires an individual piece of equipment.

Meaningful aggregations at a broader level would require information about the layout of the production line and the production schedule for different harnesses. Again, this is beyond the scope of the prototype version of HARNESS PLANEX.

7.1.2 *Choice of Technologies and Manufacturing Methods*

During the planning process, the only technology choice decisions made by HARNESS PLANEX concern the type of machine selected for the activity (e.g., a "cs-26" or a "crimper" for wire cutting). No considerations of the *number* of machines required are made as only one machine is used for each activity. As a result, the technology selection process is relatively simple. Again, an extension to include job shop scheduling would introduce more complexities.

With respect to the *extreme* activities (e.g., the tinning of wire ends), technology choices are determined solely as a function of the information stored in the associated *terminal-location* object. However, for a wire cutting activity associated with a wire body, the process is more complex because some cutting machines can cut and apply terminals to wire ends while others cannot. HARNESS PLANEX selects the type of cutting machine using two steps:

- Step 1.* An appropriate cutting machine is proposed for each end of a wire on the basis of its type of terminal.
- Step 2.* The two proposals are examined and a cutting machine is selected for the wire body.

The principal limitation of the technology choice model is that it is local to each harness. No considerations of the current machine work load, the layout of the production line or the production volumes of harnesses are incorporated into the analysis. The nature of the harness industry is so dynamic (e.g., the volumes of some part numbers may change every week) that a system which considered all factors would be very complex. Therefore, the technology selection process used by HARNESS PLANEX mimics the process employed by the harness manufacturer's production engineers.

7.1.3 *Estimation of Activity Durations and Costs*

HARNESS PLANEX estimates activity durations and costs using average productivities and unit costs provided by the industrial engineering department of the harness manufacturer. These values are stored in the duration knowledge sources of the knowledge base. Some of these relationships are a function of wire characteristics (e.g., the wire length for a cutting activity) while others involve considerations of harness topology (e.g., the number of wires linked to a particular terminal for the *tin*, *splice* or *molding* activities).

There are some limitations with respect to how HARNESS PLANEX estimates activity durations:

- no variability measure is provided for the expected activity durations; and
- no analysis is done with respect to the factors affecting activity durations.

However, these limitations reflect the *status quo* of the current manual means of generating process sheets. A richer model for activity duration computations may provide some advantages, but it must first be developed by the manufacturer.

With respect to manufacturing costs, HARNESS PLANEX uses an *average minute cost* provided by the manufacturer's accounting department. This unit cost factor is determined by analyzing information related to the manufacturing process such as inventory costs, usage of resources (labor, equipment and materials) and other indirect costs. Modification of this estimating model would be desirable as some of the cost elements are not directly related to the characteristics of individual harnesses. Allocated cost elements often make average costs derived from accounting information inaccurate [56].

7.2 System Architecture

The implementation of HARNESS PLANEX uses the four components of the PLANEX architecture:

1. *representational structures* describe the harness and manufacturing process;
2. *domain operators* perform manufacturing planning tasks;
3. *knowledge sources* detail harness manufacturing knowledge used by the operators; and
4. *user interface mechanisms* control the problem-solving process.

7.2.1 Representational Structures

Figure 7-3 shows the different objects used in HARNESS PLANEX to represent a harness and the activities required for its manufacture. As described in the previous section, *wire* objects are decomposed into two objects corresponding to the left and right end (*extreme*) of the wire and a third object describing the *body* of the wire. Each wire extreme is associated with a single *terminal-location* object that specifies the type of terminal and its location in the harness. *Terminal-location* objects are used to represent connections between wires. Activities associated with a wire extreme, such as manual application of terminals, splicing, tinning and molding are all linked to particular *terminal-location* objects. Other activities such as wire cutting are linked to *body* objects. On top of the hierarchy are *subassembly* objects linking one or more *wire* objects.

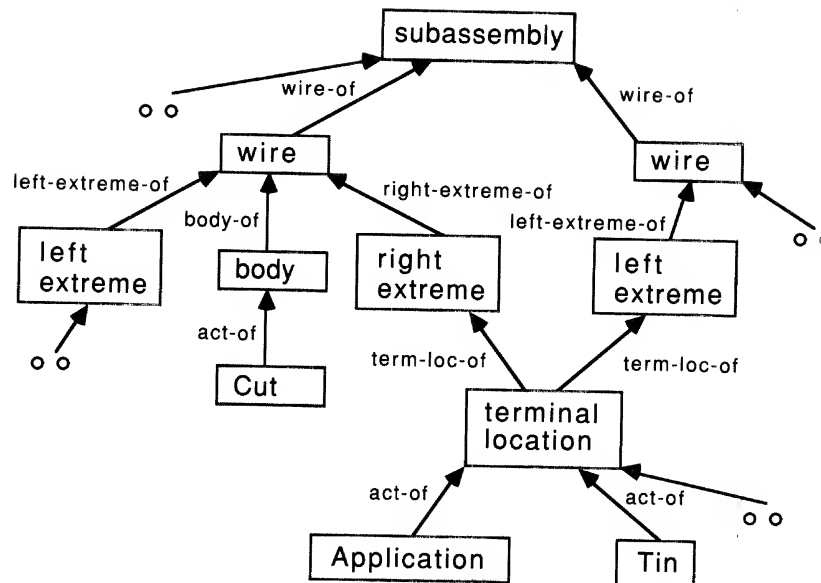


Figure 7-3. Representational Structures Used in HARNESS PLANEX

7.2.2 Domain Operators

HARNESS PLANEX generates process sheets by applying five types of problem-solving operators:

- *Wire* operators are applied to wire objects to generate objects of the harness model. Examples of wire operators are those which create body and extreme objects, and which aggregate wires into subassemblies.
- *Body* operators are applied to body objects to generate activities to manufacture a wire body.
- *Extreme* operators are applied to terminal-location objects. Examples are the operators that create manufacturing activities for a wire end, and those that determine if cutting the wire and applying the terminal can be performed simultaneously.
- *Activity* operators are applied to activity objects to select manufacturing technologies and estimate activity durations.
- *Machine* operators are applied to machine objects to compute the total usage of a machine.

The following discussion describes the behavior of the individual operators, the *Domain Operator Schemas* (DOSs) and the types of *Knowledge Sources* (KSs) used by the operators.

7.2.2.1 Wire Operators HARNESS PLANEX has six wire operators:

- *Create-Wire-Body* creates an object to represent the central portion of a wire;
- *Create-Left-Extreme* and *Create-Right-Extreme* create objects to represent wire ends;
- *Delete-Left-Extreme* and *Delete-Right-Extreme* delete objects representing wire ends; and
- *Create-Subassemblies* groups wires having common terminal-location pairs into subassemblies.

Create-Wire-Body. The *Create-Wire-Body* operator is used to create the body object of a particular wire. The operator is purely algorithmic and does not require the evaluation of a knowledge source. Figure 7-4 shows the procedural code of this operator. The operator is applied to a wire object and performs three steps:

- Step 1. Generate the name of the body object as *<wire-name>-body*, where *<wire-name>* is the name of the wire object to which the operator is applied.
- Step 2. If the body object does not exist: create it.
- Step 3. Link the body object to the wire object.

The preconditions and effects of the *Create-Wire-Body* operator are shown in the *Domain Operator Schema* (DOS) of Figure 7-5. The operator requires that the *is-a* slot of the wire object is filled, verifying that it is a wire object, and that the *name* of the wire is defined. The operator stores the name of the body object in the *has-body* slot of the wire object and fills the *is-a* and *body-of* slots in the newly created object. The three effects are all predictable because the names of the schemas and slots where information is stored are known before the operator is executed.

```
(defun create-wire-body (wire)
  (let* ((name (get-value wire 'name)) ; gets the name of the wire
        (name-body (append-atom name '-body)) ; appends -body
        (is-a (get-value wire 'is-a)))
    (cond ((equal is-a 'wire) ; check that it is a wire object
          (delete-schema name-body) ; delete previous schema
          (cschema name-body ; macro for creating a schema
                   ('is-a 'body)
                   ('name name-body)
                   ('body-of wire)
                   ('has-acts))))))
```

Figure 7-4. Procedural Code of the *Create-Wire-Body* Operator

```

(defschema Create-Wire-Body
  (is-a          operator)
  (domain-type   wire)
  (application-object current-object)
  (input-objects current-object current-object)
  (input-slots    is-a name)
  (input-bindings nil <wire-name>)
  (input-cond-types filled filled)
  (output-objects current-object <wire-name>-body
                  <wire-name>-body)
  (output-slots   has-body is-a body-of)
  (output-bindings nil nil nil)
  (output-predictable yes yes yes)
  (output-effect-type fill fill fill))

```

Figure 7-5. Domain Operator Schema for the *Create-Wire-Body* Operator

Create-Left-Extreme and *Create-Right-Extreme*. The *Create-Left-Extreme* and *Create-Right-Extreme* operators are similar to the *Create-Wire-Body* operator described above. Both are algorithmic and each creates an object to represent one of the two ends of a wire. These objects are titled *<wire-name>-left* and *<wire-name>-right*, where *<wire-name>* is the name of the wire. The operator performs four steps:

- Step 1.* Generate the names of the extreme objects from the name of the body object (e.g., *<wire-name>-left*).
- Step 2.* If the extreme object does not exist: create it.
- Step 3.* Create terminal-location objects named *<term-type>-<location>*, where *<term-type>* is the type of terminal on a wire end and *<location>* is its corresponding location in the harness.
- Step 4.* Link all objects into the representational structures.

Figure 7-6 shows the DOS of the *Create-Left-Extreme* operator. The operator uses as input the name (*name*), terminal type (*term-left*) and location (*loc-left*) of the wire object to which the operator is applied. It has the explicit requirement that the *has-left-extreme* slot of the wire object be "erased" before the operator is executed. This precondition is used by the control operators when a change to the harness design is made. The "erased" precondition will cause HARNESS PLANEX to include *delete* operators in the agenda to delete old information before new results are generated. The results are stored in the wire object (linked to the extreme object), the created extreme object (type and location) and the links to the terminal object.

```

(defschema Create-Left-Extreme
  (is-a          operator)
  (domain-type   wire)
  (application-object current-object)
  (input-objects current-object current-object
                 current-object current-object)

  (input-slots   is-a name term-left loc-left
                 has-left-extreme)
  (input-bindings nil <wire-name> <term-left> <loc-left>
                 nil)
  (input-cond-types filled filled filled filled erased)
  (output-objects current-object <wire-name>-left
                 <wire-name>-left
                 <term-left>-<loc-left> term-location)

  (output-slots   has-left-extreme is-a has-term-loc
                 is-a is-a+inv)
  (output-bindings nil nil nil nil nil)
  (output-predictable yes yes yes yes yes)
  (output-effect-type fill fill fill fill fill))

```

Figure 7-6. Domain Operator Schema for the *Create-Left-Extreme* Operator

Delete-Left-Extreme and *Delete-Right-Extreme*. The *Delete-Left-Extreme* and *Delete-Right-Extreme* operators delete wire end and terminal-location objects associated with a given wire. They are used whenever there are changes to the type of terminals in the harness. The operator:

- Step 1. Identifies the objects to be deleted.
- Step 2. Deletes the appropriate schemas.
- Step 3. Changes the relational links as appropriate.

The DOS of the *Delete-Left-Extreme* operator is shown in Figure 7-7. Inputs assure that wire end and terminal-location objects exist. The outputs are the objects deleted by the operator. In contrast to the operators that create objects, the name of the schemas to be deleted are not formed from the type of terminal at a particular wire end but are retrieved from the *has-left-extreme* and *has-term-loc* slots of the corresponding wire and wire end objects. HARNESS PLANEX must delete the objects associated with the *old* type of terminal (still stored in the extreme when the type has changed) which is different from the current value stored in the *term-left* slot of the *wire* schema.

Create-Subassemblies. During the manufacturing process of a harness, shop floor control is performed on groups of wires called *subassemblies*. Each sub-assembly is composed of one or more wires having common terminal-location pairs. In HARNESS PLANEX, aggregation of wires into subassemblies is performed by the *Create-Subassemblies* operator. The algorithm for this operator is

```

(defschema Delete-Left-Extreme
  (is-a          operator)
  (domain-type   wire)
  (application-object current-object)
  (input-objects current-object current-object
                 current-object)
  (input-slots   is-a term-left has-left-extreme)
  (input-bindings nil nil nil)
  (input-cond-types filled filled filled)
  (output-objects current-object <left-ext>
                 <left-term-loc>)
  (output-slots   has-left-extreme has-term-loc is-a)
  (output-bindings <left-ext> <left-term-loc> nil)
  (output-predictable yes yes yes)
  (output-effect-type erase erase erase))

```

Figure 7-7. Domain Operator Schema for the *Delete-Left-Extreme* Operator

shown below. The operator is more complex than other operators of the system because it must search through the representational structures for all wires objects connected through common terminal-locations.

Step 1. Initialize

- 1.1 Let *all-term-locs* be the set of all terminal-location pairs in the context.
Let *coupled-term-locs* be an empty list.

Step 2. Analyze Uncoupled Terminal-Location Objects

- 2.1 Let *uncoupled-term-locs* \leftarrow *all-term-locs* \setminus *coupled-term-locs*.
If *uncoupled-term-locs* is empty, stop.
Let *ini-term-loc* be the first element of *uncoupled-term-locs*.
Let *coupling-term-locs* \leftarrow {*ini-term-loc*}.
Let *wire-list* be an empty list.
- 2.2 If *coupling-term-locs* is empty: go to Step 3. Let *term-loc* be the first element of *coupling-term-locs*.
Let *coupling-term-locs* \leftarrow *coupling-term-locs* \setminus {*term-loc*}.
Let *coupled-term-locs* \leftarrow *coupled-term-locs* \cup {*term-loc*}.
Let *extreme-list* be the set of wire end objects linked to *term-loc* that have not yet been analyzed.
- 2.3 If *extreme-list* is empty: go to Step 2.2.
Let *extreme* be the first element of *extreme-list*.
Let *extreme-list* \leftarrow *extreme-list* \setminus {*extreme*}.
- 2.4 Get the *new-wire* object linked to *extreme*.
Let *other-extreme* and *other-term-loc* be the objects linked to the other end of *new-wire*.

- 2.5 Let $wire-list \leftarrow wire-list \cup \{new-wire\}$.
Let $extreme-list \leftarrow extreme-list \cup \{other-extreme\}$.
- 2.6 Let $coupling-term-loc \leftarrow coupling-term-loc \cup \{other-term-loc\}$.
Go to Step 2.3.

Step 3. Create Subassembly

- 3.1 Create a subassembly object with a *has-wires* slot that stores the value of *wire-list*.
Go to Step 2.1.

The set of subassemblies produced by the *Create-Subassemblies* operator depends on the topology of the complete harness. Each time a terminal-location object is created or deleted, the *Create-Subassemblies* operator should be executed to compute the set of subassemblies.

The DOS for the operator is presented in Figure 7-8. The *is-a+inv* input slot is used to insert this operator in the agenda when the terminal-location object is changed (e.g., it is "filled" with a new value). Thus, the operator is executed whenever a terminal object is created. As indicated by the DOS, the *Create-Subassemblies* operator is not applied to a particular object. It accesses the complete set of terminal-location objects from the *terminal-location* frame. The output slots describe the new subassembly. The results are unpredictable as the wires which comprise the subassembly are not known until the operator is executed.

7.2.2.2 Body Operators HARNESS PLANEX has only one body operator named *Create-Activities-Body*. This operator is applied to the body object created by the *Create-Wire-Body* operator and computes the set of manufacturing activities required for the wire body using the *KS-Activities-Body* knowledge source (see p. 272). In the current version of the system, there is only one type of body activity used to cut wires. The architecture allows other activities applied to wire bodies to be included (e.g., painting the wire insulation). The operator performs three steps:

- Step 1.* Evaluate the *KS-Activity-Body* KS and generate the cut activity.
- Step 2.* Create the activity schema.
- Step 3.* Link the activity to the associated object.

Activity objects are linked to the wire body object using the *has-acts* and *act-of* relationships, as indicated in the DOS shown in Figure 7-9. The schema also indicates that the generated list of activities is not predictable because the names of the activities are not known until the operator has evaluated the *KS-Activities-Body* knowledge source.


```
(defschema Create-Subassemblies
  (is-a          operator)
  (domain-type   nil)
  (application-object nil)
  (input-objects term-location)
  (input-slots   is-a+inv)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects subassembly <subassem> <wires>)
  (output-slots   is-a+inv has-wires wire-of)
  (output-bindings <subassem> <wires> nil)
  (output-predictable yes no no)
  (output-effect-type fill fill fill))
```

Figure 7-8. Domain Operator Schema for the *Create-Subassemblies* Operator

```
(defschema Create-Activities-Body
  (is-a          operator)
  (domain-type   body)
  (application-object current-object)
  (input-objects  current-object)
  (input-slots   is-a)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object <acts>)
  (output-slots   has-acts is-a)
  (output-bindings <acts> nil)
  (output-predictable yes no)
  (output-effect-type fill fill))
```

Figure 7-9. Domain Operator Schema for the *Create-Activities-Body* Operator

7.2.2.3 Extreme Operators HARNESS PLANEX includes four operators which are applied to the terminal-location objects associated with particular wire ends:

- *Get-Peeling* determines the appropriate length of insulation to peel from a wire end;
- *Get-Cut-Machine* recommends an appropriate cutting machine for a wire on the basis of the type of terminal attached to one of its ends;
- *Create-Activities-Extreme* creates objects representing the manufacturing activities associated with a set of wire ends; and
- *Delete-Activities-Extreme* deletes wire end manufacturing activity objects.

Get-Peeling. The *Get-Peeling* operator computes the appropriate length of insulation to peel from the wire ends linked to a terminal-location object using the *KS-Peeling* knowledge source (see p. 272). Two steps are performed:

```
(defschema Get-Peeling
  (is-a          operator)
  (domain-type   term-location)
  (application-object current-object)
  (input-objects current-object)
  (input-slots   is-a)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object)
  (output-slots   peeling)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 7-10. Domain Operator Schema for the *Get-Peeling* Operator

- Step 1. Evaluate the *KS-Peeling* KS to determine the peeling length.
- Step 2. Store the results in the wire terminal-location object to which the operator is applied.

Figure 7-10 shows the DOS that describes the operator. The *Get-Peeling* operator is applied to a single terminal-location object whose *is-a* slot is "filled", indicating that it exists. The operator stores the appropriate peeling length in the *peeling* slot of this object.

Get-Cut-Machine. The appropriate machine for cutting wires is selected by the *Get-Cut-Machine* operator. The appropriate machine is determined by considering the type of terminal attached to a terminal-location object. The operator performs two steps:

- Step 1. Evaluate the *KS-Cut-Machine* KS (see p. 273) to select the cutting machine.
- Step 2. Store the result in the schema of the terminal-location object to which the operator is applied.

Figure 7-11 shows the DOS describing the *Get-Cut-Machine* operator. The operator requires the *is-a* slot of the terminal-location object be "filled", and it stores the recommended cutting machine in the *cut-machine* slot of this object.

Create-Activities-Extreme. The *Create-Activities-Extreme* operator generates the activity objects required to manufacture the set of wire ends linked to a specific terminal-location object. These activities are generated by a single *KS-Activities-Extreme* knowledge source (see p. 273). Three steps are performed:

- Step 1. Evaluate the *KS-Activities-Extreme* KS to determine the activities needed to manufacture the wire end.

```
(defschema Get-Cut-Machine
  (is-a          operator)
  (domain-type   term-location)
  (application-object current-object)
  (input-objects current-object)
  (input-slots   is-a)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object)
  (output-slots   cut-machine)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 7-11. Domain Operator Schema for the *Get-Cut-Machine* Operator

Step 2. Create the appropriate activity schemas.

Step 3. Link the activities to the corresponding terminal-location object.

Figure 7-12 shows the DOS that describes the inputs and outputs of the operator. The operator is applied to terminal-location objects which must exist (the *is-a* slot must be “filled”) and must not have any associated activities (the *has-acts* slot must be “empty”). This insures that the operator will be reinvoked whenever the harness configuration changes. The DOS indicates that the name of the created activity objects are not known until the operator is executed.

Delete-Activities-Extreme. The *Delete-Activities-Extreme* operator is used to delete the activities linked to a particular terminal-location object whenever there are changes in the type of terminal at this terminal-location object. The operator does not require KS evaluation. Its three steps are:

```
(defschema Create-Activities-Extreme
  (is-a          operator)
  (domain-type   term-location)
  (application-object current-object)
  (input-objects current-object current-object)
  (input-slots   is-a has-acts)
  (input-bindings nil nil)
  (input-cond-type filled erased)
  (output-objects current-object <acts>)
  (output-slots   has-acts is-a)
  (output-bindings <acts> nil)
  (output-predictable yes no)
  (output-effect-type fill fill))
```

Figure 7-12. Domain Operator Schema for the *Create-Activities-Extreme* Operator

```
(defschema Delete-Activities-Extreme
  (is-a          operator)
  (domain-type   term-location)
  (application-object current-object)
  (input-objects current-object current-object
                 current-object)
  (input-slots   is-a has-acts terminal)
  (input-bindings nil <acts> nil)
  (input-cond-type erased filled filled)
  (output-objects current-object <acts> <machine>)
  (output-slots   has-acts technology used-by)
  (output-bindings nil <machine> nil)
  (output-predictable yes yes yes)
  (output-effect-type erase erase fill))
```

Figure 7-13. Domain Operator Schema for the *Delete-Activities-Extreme* Operator

- Step 1.* Identify the activities associated with the terminal-location object and the machines used to perform these *extreme* activities.
- Step 2.* Delete the corresponding activity schemas.
- Step 3.* Remove the activity from the list of those which are performed with the identified machine.

Figure 7-13 shows the DOS that describes the preconditions and effects of this operator. The *Delete-Activities-Extreme* operator acts on a terminal-location object whose *has-acts* slot is not empty and whose *terminal-type* slot has been filled with a new value. When it is executed, the operator erases the *has-acts* slot of the terminal-location object and changes the value of the *used-by* slot of the machine objects which had been linked to the activity objects being deleted.

7.2.2.4 Activity Operators HARNESS PLANEX has three activity operators:

- *Select-Technology-Body* selects appropriate types of machines to perform the manufacturing activities for the wire body (e.g., cutting);
- *Select-Technology-Extreme* selects machines to perform the manufacturing activities of the wire ends; and
- *Get-Duration* estimates the duration of body and extreme activities.

Select-Technology-Body. An appropriate type of machine for cutting a wire is selected by the *Select-Technology-Body* operator. The operator is applied to a body activity and examines the cutting machine choices made by the *Get-Cut-Machine* operator for the two terminal-location objects associated with the wire. To select the cutting machine, the operator performs three steps:

- Step 1.* Build the name of the *Technology* KS (see p. 273) to be evaluated by concatenating the prefix *KS-Technology-* with the name of the activity used to manufacture the body (e.g., cut).

```

(defschema Select-Technology-Body
  (is-a      operator)
  (domain-type  body-activity)
  (application-object  current-object)
  (input-objects  current-object current-object <body>
                  <wire> <wire> <left-ext> <right-ext>
                  <term-loc-left> <term-loc-right>)
  (input-slots  is-a act-of body-of has-left-extreme
                has-right-extreme has-term-loc
                has-term-loc cut-machine cut-machine)
  (input-bindings  nil <body> <wire> <left-ext>
                  <right-ext> <term-loc-left>
                  <term-loc-right> nil nil)
  (input-cond-types  filled filled filled filled filled
                    filled filled filled filled)
  (output-objects  current-object <machine>)
  (output-slots  technology used-by)
  (output-bindings  <machine> nil)
  (output-predictable  yes no)
  (output-effect-type  fill fill))

```

Figure 7-14. Domain Operator Schema for the *Select-Technology-Body* Operator

- Step 2. Evaluate this KS to determine which machine to use to manufacture the wire body.
- Step 3. Store the result in the activity schema and add the activity to the list of activities which use the machine.

The DOS of the operator is shown in Figure 7-14. The preconditions of the operator indicate that its output is dependent upon the values of the *cut-machine* slot of the left and right terminal-location objects of the wire. The operator has the predictable effect of storing a value in the *technology* slot of the activity and the unpredictable effect of modifying the *used-by* slot of the machine selected by the operator.

Select-Technology-Extreme. The *Select-Technology-Extreme* operator selects appropriate machines for manufacturing activities related to wire ends. Selection is done using *Technology* KSs. When applied to an *extreme* activity, the operator performs three steps:

- Step 1. Build the name of the *Technology* KS (see p. 273) to be evaluated by concatenating the prefix *KS-Technology-* with the name of the activity used to manufacture the wire extreme.
- Step 2. Evaluate this KS to determine which machine to use to manufacture the wire end.
- Step 3. Store the result in the activity object and add the activity to the list of activities which use the machine.

```
(defschema Select-Technology-Extreme
  (is-a operator)
  (domain-type extreme-activity)
  (application-object current-object)
  (input-objects current-object)
  (input-slots is-a)
  (input-bindings nil)
  (input-cond-types filled)
  (output-objects current-object <machine>)
  (output-slots technology used-by)
  (output-bindings <machine> nil)
  (output-predictable yes no)
  (output-effect-type fill fill))
```

Figure 7-15. Domain Operator Schema for the *Select-Technology-Extreme* Operator

The DOS is shown in Figure 7-15. The operator requires the *is-a* slot of the activity object to be filled, and stores the name of the selected machine in the corresponding *technology* slot. The *Select-Technology-Extreme* operator also modifies the *used-by* slot of the machine object allocated to the activity.

Get-Duration. The *Get-Duration* operator estimates the duration of body and extreme activities using *Duration* KSs. The operator is applied to either a wire body manufacturing activity or an wire end manufacturing body activity. Three steps are performed:

- Step 1.* Build the name of the *Duration* KS (see p. 275) to be evaluated by concatenating the prefix *KS-Duration-* with the type of the manufacturing activity being considered (e.g., tinning).
- Step 2.* Evaluate this KS to determine the duration of the activity.
- Step 3.* Store the result in the activity object.

The computed duration is stored in the *duration* slot of the activity object, as indicated in the DOS of Figure 7-16.

7.2.2.5 Machine Operators HARNESS PLANEX has one machine operator titled *Compute-Machine-Usage* which computes the total usage of all machines. The operator does not require KS evaluation. It is applied to all machines and sums the duration of all the activities which utilize the machine.

Figure 7-17 shows the DOS associated with the operator. The operator requires the *used-by* slot of a machine object be filled with the names of the activities using this machine, and the *duration* slots of these objects must have a value. The output of the operator is the sum of the individual activity durations and is stored in the *total-usage* slot of the machine object.

```
(defschema Get-Duration
  (is-a operator)
  (domain-type body-activity extreme-activity)
  (application-object current-object)
  (input-objects current-object)
  (input-slots is-a)
  (input-bindings nil)
  (input-cond-type filled)
  (output-objects current-object)
  (output-slots duration)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 7-16. Domain Operator Schema for the *Get-Duration* Operator

```
(defschema Compute-Machine-Usage
  (is-a operator)
  (domain-type machine)
  (application-object current-object)
  (input-objects current-object <acts>)
  (input-slots used-by duration)
  (input-bindings <acts> nil)
  (input-cond-types filled filled)
  (output-objects current-object)
  (output-slots total-usage)
  (output-bindings nil)
  (output-predictable yes)
  (output-effect-type fill))
```

Figure 7-17. Domain Operator Schema for the *Compute-Machine-Usage* Operator

7.2.3 Knowledge Sources

The knowledge base of HARNESS PLANEX contains the following types of knowledge sources:

- a *Peeling* KS estimates the length of insulation peeling (in millimeters) required for each type of terminal;
- *Activity* KSs generate the set of manufacturing activities required for body or wire extreme objects;
- *Technology* KSs select the type of cutting, molding, tinning, terminal application or splicing machine used to perform an activity; and
- *Duration* KSs estimate the expected duration of manufacturing activities.

Each type of KS is described below in detail.

```

(defschema KS-Peeling
  (is-a      ks)
  (ks-name   ks-peeling)
  (ks-type   first)
  (cond-objects current-object current-object
                current-object)
  (conditions (member terminal (6675))
              (member terminal (3227 3465 2678 2086 2769
                                2923 3242 3710 2923 2488
                                2486 3710 2490 2085 6245)))
  (lhs-rules ((T I I)
              (I T I)
              (I I T)))
  (rhs-rules ((X I I)
              (I X I)
              (I I X)))
  (actions   4 6 10))

```

Figure 7-18. KS to Determine Peeling Length

Knowledge Source for Determining Peeling Length. Before a terminal is applied to the end of a wire, a portion of the insulation must be removed. In HARNESS PLANEX, the appropriate length of insulation peeling for a specific type of terminal is determined using the *Peeling* KS. This KS is used by the *Get-Peeling* operator (see p. 265) which is applied to a terminal-location object.

The KS is shown in Figure 7-18. The type of *KS-Peeling* is "first", indicating that only the first rule whose conditions are satisfied should be fired. The KS has three rules which select a peeling length on the basis of the terminal type. One of three possible peeling lengths (4, 6 or 10 millimeters) is returned to the *Get-Peeling* operator to be stored with the terminal-location object.

Knowledge Sources for Activity Creation. HARNESS PLANEX distinguishes between two types of manufacturing activities:

- *extreme* activities are applied to the ends of a wire (e.g., tinning, welding or molding); and
- *body* activities are applied to the wire as a whole (e.g., cutting).

In the current version of the system, each wire is manufactured with only one *body* activity called *cut*. The *KS-Activity-Body* KS is used to generate a cut activity for the *Create-Activities-Body* operator (see p. 264).

A particular set of *extreme* activities is associated with manufacturing each wire end. These activities are determined by the *Create-Activities-Extreme* operator (see p. 266) which uses the *KS-Activities-Extreme* KS shown in Figure 7-19. Each rule corresponds to a particular manufacturing activity. The conditions are based on the type of terminal attached to the end of the wire. The


```

(defschema KS-Activities-Extreme
  (is-a      ks)
  (ks-name   ks-activities-extreme)
  (ks-type   all)
  (cond-objects current-object current-object current-object
                 current-object current-object)
  (conditions (member terminal (3242 2769 2678 2486))
              (member terminal (3197 3401))
              (member terminal (2490 2086 3197 3401 2486))
              (member terminal (3242))
              (member terminal (6675)))
  (lhs-rules (T I I I I)
             (I T I I I)
             (I I T I I)
             (I I I T I)
             (I I I I T))
  (rhs-rules (X I I I I)
             (I X I I I)
             (I I X I I)
             (I I I X I)
             (I I I I X))
  (actions   manual_application splice tin welding
             molding))

```

Figure 7-19. KS to Determine Extreme Activities

KS type is "all", indicating that all rules whose conditions are satisfied should be fired, and thus generating all possible activities. For example, if the terminal type of a wire end is "3242", the first and third rules fire and the results are: "(manual_application tin)", indicating that this wire end requires the application of the terminal (by hand) and tinning.

Knowledge Sources for Technology Selection. As described in Section 7.1.2, the selection of machines for manufacturing activities is a function of activity and terminal types. In the case of a cutting activity associated with each wire body, two KSs are used: one by the *Get-Cut-Machine* operator (see p. 266) and one by the *Select-Technology-Body* operator (see p. 268). HARNESS PLANEX selects an appropriate machine for manufacturing a wire end using the *Get-Cut-Machine* operator and the *KS-Cut-Machine* KS shown in Figure 7-20. Depending on the terminal type, one of two possible machines ("cs-26" or "crimper") is selected for each wire end. Then HARNESS PLANEX determines a unique cutting machine for the wire using the *Select-Technology-Body* operator and the *KS-Technology-Cut* KS of Figure 7-21. This KS indicates that a wire should be cut on a "crimper" machine only if neither of its ends can be cut on a "cs-26" machine. During the evaluation of this KS, the *Get-Left-Cut-Machine* and *Get-Right-Cut-Machine* functions are used to retrieve the selected cutting machine names from the wire extremes objects.

```
(defschema KS-Cut-Machine
  (is-a      ks)
  (ks-name   ks-cut-machine)
  (ks-type   first)
  (cond-objects current-object current-object)
  (conditions (member terminal (2488 2490 2923 2678 3227
                                2085 6675 6245 2086 3465
                                3710))
              (member terminal (3197 3401 3242 2769
                                2486))))
  (lhs-rules (T I)
             (I T))
  (rhs-rules (X I)
             (I X))
  (actions   cs-26 crimper))
```

Figure 7-20. KS to Determine the Appropriate Cutting Machine for Wire Extremes

```
(defschema KS-Technology-Cut
  (is-a      ks)
  (ks-name   ks-technology-cut)
  (ks-type   first)
  (cond-objects function function function)
  (conditions (equal (get-left-cut-machine
                      current-object) 'cs-26)
              (equal (get-right-cut-machine
                      current-object) 'cs-26)
              (equal (get-left-cut-machine
                      current-object) 'crimper)
              (equal (get-right-cut-machine
                      current-object) 'crimper))
  (lhs-rules (T I I I)
             (I T I I)
             (F F T I)
             (F F I T))
  (rhs-rules (X I)
             (X I)
             (I X)
             (I X))
  (actions   cs-26 crimper))
```

Figure 7-21. KS to Select Cutting Machines for Wires

For manufacturing activities associated with extreme objects, the technology selection process is simpler. Machines are readily selected by evaluating a KS titled *KS-Technology-⟨act-type⟩*, where *⟨act-type⟩* is the type of activity whose technology is being determined (e.g., "tin").

Knowledge Sources for Activity Durations. HARNESS PLANEX determines the expected duration of manufacturing activities using *Duration* KSs. These are used by the *Get-Duration* operator (see p. 270). Similar to *Technology* KSs, *Duration* KSs are identified with the activity name. The KS name is of the form *KS-Duration-act-type*, where *act-type* is the name of the activity whose duration is being computed. Thus, a KS titled *KS-Duration-Molding* is used to determine the expected duration of *molding* activities.

An example of a *Duration* KS is shown in Figure 7-22. This KS indicates that the duration of a tinning activity is dependent upon: (1) the type of terminal being tinned; and (2) the number of wire end objects connected to the terminal-location object. For example, if a terminal-location object type "2490" is linked to three wires, the expected duration of the activity is $0.01 \times 3 = 0.03$ minutes. Similarly, for splicing activities, the duration is a function of the number of wires linked to a particular terminal-location object. However, there are several *Duration* KSs in which the duration is independent of harness topology and is only a function of the individual wires (e.g., the duration of a cutting activity is only a function of the wire length).

```
(defschema KS-Duration-Tin
;;
;; This KS returns the typical duration of a tin activity based
;; on the number of wire ends applied on the same terminal.
;; - The first rule indicates that for ends with terminals
;;   2490, 2086 or 2486, the typical duration is 0.01 times
;;   the number of wires applied.
;; - The second rule indicates that for ends with terminals
;;   3197, 3401 or 3242, the typical duration is 0.02 times
;;   the number of wires applied.
;;
  (is-a      ks)
  (KS-name   KS-duration-tin)
  (KS-type   first)
  (cond-objects current-object current-object function)
  (conditions (member terminal (2490 2086 2486))
              (member terminal (3197 3401 3242))
              (equal (length (get-values 'current-object
                                         'term-loc-of)) <no-wires>))

  (lhs-rules (T F T)
             (F T T))
  (rhs-rules (X I)
             (I X))
  (actions  (* <no-wires> 0.01)
            (* <no-wires> 0.02)))
```

Figure 7-22. Example of a Duration KS Used in HARNESS PLANEX

7.2.4 User Interface Mechanisms

HARNESS PLANEX incorporates some of the user interaction mechanisms of PLANEX described in Section 4.4, such as command menus and questions to the user. The KNOWLEDGE SOURCE ACQUISITION MODULE was used to build the knowledge base. HARNESS PLANEX does not include any type of graphical schedule display since precedences among the manufacturing activities are not determined.

Figure 7-23 shows the command menus of the HARNESS PLANEX system. Much of the menu structure parallels that of CONSTRUCTION PLANEX (see Section 6.2.4). With these menus, the user may control the execution of the system in any of the three levels of execution discussed in Section 3.1.4:

- *Strategic.* The user may invoke the control operators of PLANEX by entering the CONTROL PANEL (CP) from the *Operations* menu. With the CP, the user may create and modify sequences of domain operators. The strategic level is useful for propagating plan changes (e.g., a change in the design of the harness).
- *Operative.* The user may execute domain operators using the *Individual Operations* menu. This menu contains a list of all of the operators described in the previous section. When the user selects an operator for execution, HARNESS PLANEX verifies whether its preconditions are satisfied. If they are, the operator is executed and control is returned to the user. If not, the system asks if the unsatisfied preconditions should be inserted as goals in the agenda.
- *Interface.* The user may display results or ask for explanations of planning decisions using the *Display* menu and the *Explain* menu. Output reports (e.g., process sheets) are produced via the *Report* menu.

The *Change* menu provides a tool for modifying plan information and inserting changes in the agenda. When the user selects this menu, HARNESS PLANEX asks the user to specify the type of object to which the change is applied (e.g., wire, extreme, terminal-location or activity). Then one of the submenus is displayed (e.g., *Wire*, *Extreme*, *Term-Loc* or *Activity* menu, the lowest-level submenus of Figure 7-23) to select which objects will be changed. Similar to CONSTRUCTION PLANEX, these menus work in conjunction with the other menus of the system to select objects. For example, if the user wants to estimate the duration of manufacturing activities, the *Activity* menu is displayed to let the user select those activities whose duration will be computed.

Results of the planning process are presented in three types of reports:

- *process sheet* reports describe the manufacturing activities required for each wire and the wires comprising each subassembly;
- *time sheet* reports indicate the technology choice and expected duration of the manufacturing activities; and

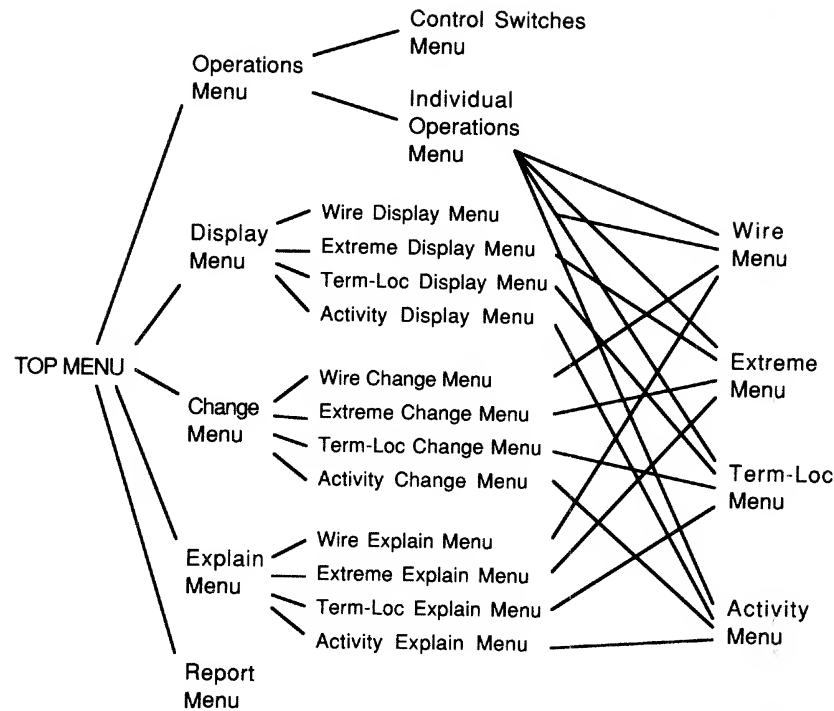


Figure 7-23. Menus of the HARNESS PLANEX System

- *machine usage* reports indicate the total usage of each machine utilized in manufacturing the harness.

The format of these reports is similar to those currently used by the harness manufacturer.

An example of a process sheet report produced by HARNESS PLANEX is shown in Figure 7-24. In this report, subassemblies are sorted by the number of wires they contain, and wires in a subassembly are sorted by length. Each row corresponds to a wire and its associated extreme, body and terminal-location and describes the wire from left to right (left-terminal, body, right-terminal). For example, wire "9E" has a left end linked with terminal "3401" that is common (has a "U" in column "U") to the left end of wire "9F" and to the right end of wires "9A", "9C" and "9B". On its right end, wire "9E" has a terminal "2086" that may be applied when cutting the wire (indicated with an "X" in column "M") and requires tinning (indicated with a "T" in column "T"). Column "-" indicates that a terminal has already appeared in a previous row of the process sheet. Column "PEL" indicates how much insulation to peel from the wire end. Column "CABLE" specifies the wire color, gauge and insulation type. Column "CUT" indicates the length of a wire (in millimeters).

SUB	T	TERM-UM	PEL	CABLE	WIRE	CUT	PEL	TERM-UM	T
1	T	3401 U	10	951811	9E	1875.0	6	2086 X	T
1	T	3401-U	10	802011	9F	1655.0	6	6245 X	
1		3710 X	6	952011	9A	950.0	10	3401-U	T
1	T	2086 X	6	951811	9C	640.0	10	3401-U	T
1		6245 X	6	802011	9B	415.0	10	3401-U	T
8	T	2490 X	6	971600	150L	1215.0	6	3242	W
8	T	3197 U	10	951800	150E	510.0	6	2085 X	
8	T	3197-U	10	952000	150C	390.0	6	2923 X	
8	T	3197-U	10	971400	150A	390.0	6	3242-	W
8	T	3197-U	10	952000	150B	210.0	6	2923 X	
10		3465 X	6	971600	151H	825.0	6	3242	W
10	T	3401 U	10	951800	151E	530.0	6	2085 X	
10	T	3401-U	10	971800	151A	420.0	6	3242-	W
10	T	3401-U	10	952000	151C	390.0	6	2923 X	
10	T	3401-U	10	952000	151B	210.0	6	2923 X	
5		2678 X	6	951815	15A	2075.0	10	3401 U	T
5	T	3401-U	10	951815	15B	750.0	6	2086 X	T
5	T	3401-U	10	802015	15C	530.0	6	6245 X	
6	T	3197 U	10	951605	29C	1550.0	6	3465 X	
6		2488 X	6	951605	29A	1055.0	10	3197-U	T
6	T	3197-U	10	951605	29B	640.0	6	3465 X	
4		2678 X	6	951816	14A	910.0	10	3401 U	T
4	T	3401-U	10	951816	14C	680.0	6	2086 X	T
4		6245 X	6	802016	14B	455.0	10	3401-U	T
3		2769	6	951601	12B	2605.0	6	3227 X	
3		2769-	6	951601	12A	1215.0	6	3227 X	
2		2769	6	951606	11B	2505.0	6	3227 X	
2		2769-	6	951606	11A	1215.0	6	3227 X	
7		2678 X	6	801804	68	2605.0	4	6675 X	
9	T	2486	6	971600	150H	825.0	6	3465 X	
12		2678 X	6	951602	228	750.0	6	3465 X	
11		2678 X	6	951616	227	750.0	6	3465 X	

Figure 7-24. Example of a Process Sheet Report Produced by HARNESS PLANEX

An example of a time sheet report produced by HARNESS PLANEX is shown in Figure 7-25. Output is ordered as in the process sheet report. Again, information corresponds to a left-to-right description of the harness. The column designations are as follows: "SUB" indicates subassembly number; "MOLD-D" and "MOLD-M" indicate the duration and machine for molding activities; "APPL-D" and "APPL-M" indicate duration and machine for connector application; and "CUT-D" and "CUT-M" indicate duration and machine for wire cutting. The example report indicates that wire "9A" will be cut on machine "cs-26" and that this operation will take an average of 0.025 minutes. Similarly, the activity of tinning terminal "3401" at the left end of wires "9F" and "9E" and right end of wires "9A", "9C" and "9B" will take 0.06 minutes and does not require any machine (it will be done by hand using a tinning tub).

SUB	MOLD-D APPL-D	MOLD-M APPL-M	TIN/W TIN/W	APPL-D MOLD-D	APPL-M MOLD-M	CUT-D	CUT-M	WIRE
1			0.060	0.251	SC-3401	0.025	CS-26	9E
1			0.010					
1						0.025	CS-26	9F
1						0.025	CS-26	9A
1			0.010			0.017	CS-26	9C
1						0.011	CS-26	9B
8	0.126	PACK-US	0.010			0.025	CS-26	150L
8			0.048	0.256	SC-3197	0.017	CS-26	150E
8						0.011	CS-26	150C
8						0.011	CRIMPER	150A
8						0.011	CS-26	150B
10	0.126	PACK-US	0.024			0.017	CS-26	151H
10			0.048	0.251	SC-3401	0.017	CS-26	151E
10						0.011	CRIMPER	151A
10						0.011	CS-26	151C
10						0.011	CS-26	151B
5	0.144	SC-3401	0.036	0.032	PACK-SS	0.025	CS-26	15A
5			0.010			0.017	CS-26	15B
5						0.017	CS-26	15C
6			0.036	0.144	SC-3197	0.025	CS-26	29C
6						0.025	CS-26	29A
6						0.017	CS-26	29B
4	0.144	SC-3401	0.036	0.032	PACK-SS	0.017	CS-26	14A
4			0.010			0.017	CS-26	14C

Figure 7-25. Example of a Time Sheet Report Produced by HARNESS PLANEX

In this case, the complete set of activities are not specified, but must be inferred from the harness description in the process sheet.

7.3 Example Problem

This section describes in detail the use of HARNESS PLANEX via the example harness of Figure 7-26. This simple harness is composed of three wires named "A", "B" and "C"; each has the length, color and gauge described in Figure 7-27. The example contains two parts: (1) the system obtains an initial process plan for harness manufacture using the set of domain operators presented in the previous section; and (2) the user makes changes to the initial process plan and HARNESS PLANEX propagates these changes with the control operators of the system.

7.3.1 Obtaining an Initial Process Plan

At the start of the planning process, HARNESS PLANEX is given a file containing the information shown in Figure 7-27 and creates three wire schemas containing all the information needed to create the other objects that are used to represent the design information. The system creates these other objects using the following operators:

- *Create-Wire-Body* creates body objects (e.g., *wire-c-body*) and links these to the corresponding wire objects using *body-of* and *has-body* relationships.
- *Create-Left-Extreme* creates objects representing left ends of wires (e.g., *wire-c-left*) and links these to the wire objects using *left-extreme-of* and *has-left-extreme* relationships. This operator links the extreme objects to their corresponding terminal-location objects (e.g., *3197-2* for *wire-c-left*) using *term-loc-of* and *has-term-loc* relationships. If the terminal-location object is not present, it is created.
- *Create-Right-Extreme* creates objects representing the right end of the wires and links these to the wire objects using *right-extreme-of* and *has-right-extreme* relationships. Similarly to the *Create-Left-Extreme* operator, this operator creates and links terminal-location objects to the extreme objects.
- *Create-Subassemblies* aggregates wires into subassemblies. For the example harness, one subassembly (*sub-1*) is created.

After the objects representing the harness design information have been created, HARNESS PLANEX generates the set of manufacturing activities required for each harness component using two operators:

- *Create-Activities-Body* creates the objects representing the cutting activity for with each wire (e.g., *cut-wire-c*) and links them to the body objects using the *act-of* and *has-acts* relationships.
-

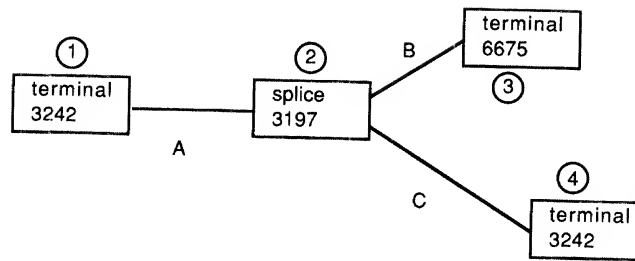


Figure 7-26. Example of a Simple Harness

HARNESS DESCRIPTION								
Wire	Gauge	Color	Plastic	Length	Loc Left	Loc Right	Term Left	Term Right
A	0.5	brown	95	950.0	1	2	3242	3197
B	0.8	tan	95	640.0	2	3	3197	6675
C	0.8	black	95	1730.0	2	4	3197	3242

Figure 7-27. Design Information for the Example Harness

- *Create-Activities-Extreme* creates the objects representing the manufacturing activities associated with each terminal-location object. For example, this operator creates the *splice-3197-2* and *tin-3197-2* activities for the 3197-2 terminal-location. Activity objects are linked to terminal-location objects using *act-of* and *has-acts* relationships.

Figure 7-28 shows the final set of representational structures created by the system. The harness has one subassembly, three wires and four terminal-location objects. Harness manufacture requires nine activities:

- three cutting activities (one for each wire);
- two activities for terminal "3242" at left end of wire "A" (manual connector application and welding);
- two activities for splice "3197" joining the right end of wire "A" to the left ends of wires "B" and "C" (splice and tin);
- one molding activity for terminal "6675" associated with the right end of wire "B"; and
- two activities for terminal "3242" at the right end of wire "C" (manual connector application and welding).

HARNESS PLANEX completes the initial plan by applying the following operators:

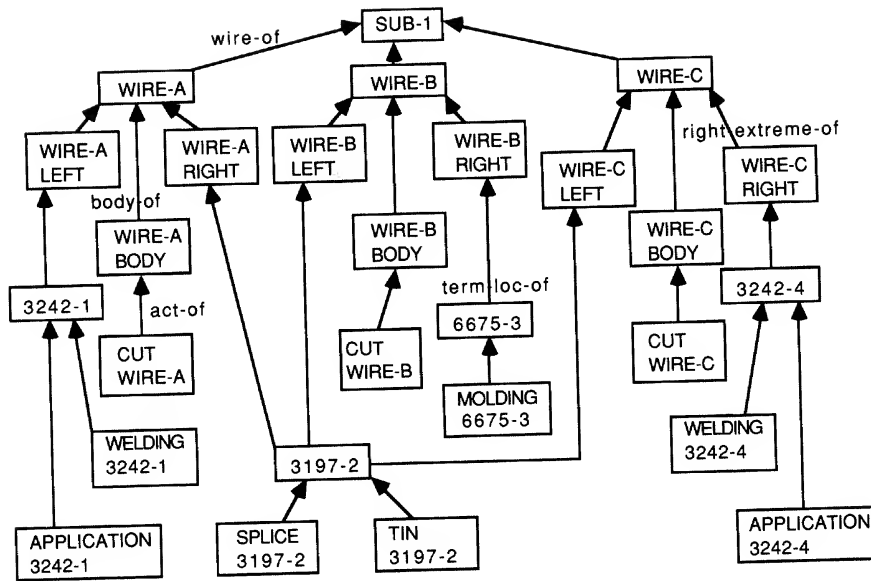


Figure 7-28. Representational Structures for the Example Harness

- *Get-Cut-Machine* determines appropriate cutting machines for terminal-location objects using the *KS-Cut-Machine* knowledge source. HARNESS PLANEX recommends machine "crimper" for cutting wire "A" on the basis of the terminal type attached to object 3242-1.
- *Select-Technology-Body* analyzes the results of *Get-Cut-Machine* and chooses a machine to cut each of the three wires using the *KS-Technology-Cut* knowledge source. The system determines that wire "C" will be cut using machine "cs-26" as neither of its ends require an operation performed by a different type of machine.
- *Select-Technology-Extreme* selects the type of machine used for each manufacturing activity on wire ends. This information is generated by *Technology KSs*. HARNESS PLANEX, using the *KS-Technology-Molding* knowledge source, determines that molding for terminal "6675" located at the right end of wire "B" will be done with machine "usm-1".
- *Get-Duration* estimates the duration of manufacturing activities. The duration of the activity *tin-3197-2* is estimated to be $0.02 \times 3 = 0.06$ minutes because three wire ends are linked to the terminal-location 3197-2. This value was obtained using the *KS-Duration-Tin* knowledge source.

The results of the planning process are displayed in Figures 7-29 to 7-31 (the organization of the reports follows that described in Section 7.2.4). Figure 7-29 shows the process sheet for the harness. The report has three rows, each

SUB	T	TERM-UM	PEL	CABLE	WIRE	CUT PEL	TERM-UM	T
1	W	3242	6	952011	A	950.0	10 3197 U	T
1	T	3197-U	10	951801	B	640.0	4 6675 X	
1	T	3197-U	10	951800	C	1730.0	6 3242	W

Figure 7-29. Process Sheet for the Example Harness

SUB	MOLD-D	MOLD-M	TIN/W	APPL-D	APPL-M	CUT-D	CUT-M	WIRE
	APPL-D	APPL-M	TIN/W	MOLD-D	MOLD-M			
1			0.012	0.126	PACK-US	0.011	CRIMPER	A
	0.144	SC-3197	0.06					
1				0.037	USM-1	0.017	CS-26	B
						0.011	CRIMPER	C
1	0.126	PACK-US	0.012					

Figure 7-30. Time Sheet for the Example Harness

MACHINE	TIME
CRIMPER	0.022
CS-26	0.017
PACK-US	0.252
SC-3197	0.144
TIN	0.060
WELDING	0.024
USM-1	0.037
TOTAL >	0.556

Figure 7-31. Machine Report for the Example Harness

describing one of the wires of the harness. For example, the first row indicates that wire "A" has manual terminal application and welding on its left end (a blank and a "W" in columns "M" and "T") and splicing and tinning on the right end (a "T" in column "T"). Figure 7-30 shows a time sheet report with the duration and the type of machines used for each manufacturing activity. For example, splicing at the right end of wire "A" and the left end of wires "B" and "C" takes 0.144 minutes, and is done with machine "sc-3197". Figure 7-31 shows the total usage (in minutes) for each type of machine and the total time required to manufacture the harness.

7.3.2 Modifications to the Process Plan

If there is a change in the design specifications of the harness, HARNESS PLANEX must modify the initial process plan to account for this change. The control operators of the PLANEX architecture may be used to modify the plan. As an example, assume that the terminal on the right end of wire "C" is changed from type "3242" to type "6675". This new terminal type is stored in the *wire-c* object by utilizing the *Wire Changes* menu of the interface. After the change is introduced by the user, the *context-chgs* slot of the agenda includes "(wire-c term-right filled)". The user may execute the *Forward Propagation Operator* (FPO) to propagate the consequences of the change, or use the *Operations* menu to manually invoke specific planning operators. Assume that the FPO operator is invoked. The system creates the network of operators and conditions shown in Figure 7-32 in six steps:

- Step 1. The "(wire-c term-right filled)" change affects the preconditions of the *Create-Right-Extreme* and *Delete-Right-Extreme* operators. Only the *Delete-Right-Extreme* operator has all of its preconditions satisfied, and it is added to the *operator-queue* slot of the agenda. The predictable effects of this operator are added to the *context-chgs* slot of the agenda.
- Step 2. The "(wire-c has-right-extreme erased)" change then activates the *Create-Right-Extreme* operator. All of the preconditions of the operator are satisfied, so it is added to the *operator-queue* and its changes are inserted into the agenda.
- Step 3. The "(3242-4 is-a erased)" change next activates the *Delete-Activities-Extreme* operator and changes affecting those machines used by the activities linked to the 3242-4 terminal-location object are made to the agenda.
- Step 4. The "(terminal-location is-a+inv filled)" change that is introduced by *Create-Right-Extreme* operator activates the *Create-Subassembly* operator and the FPO adds this operator to the *operator-queue*.
- Step 5. The "(pack-us used-by filled)" change then activates the *Compute-Machine-Usage* operator and it is inserted in the agenda.
- Step 6. Finally, the "(6675-4 is-a filled)" change activates the *Create-Activities-Extreme* operator and the predictable effect "(6675-4 has-acts filled)" is made to the agenda.

The FPO continues analyzing the remaining changes in the *context-chgs* slot of the agenda, but no new operators are activated. The result is the set of operators and conditions shown in Figure 7-32. HARNESS PLANEX interprets this network using the *Network Interpretation Operator* (NIO). Figure 7-33 presents the resulting network of operators. This network shows that:

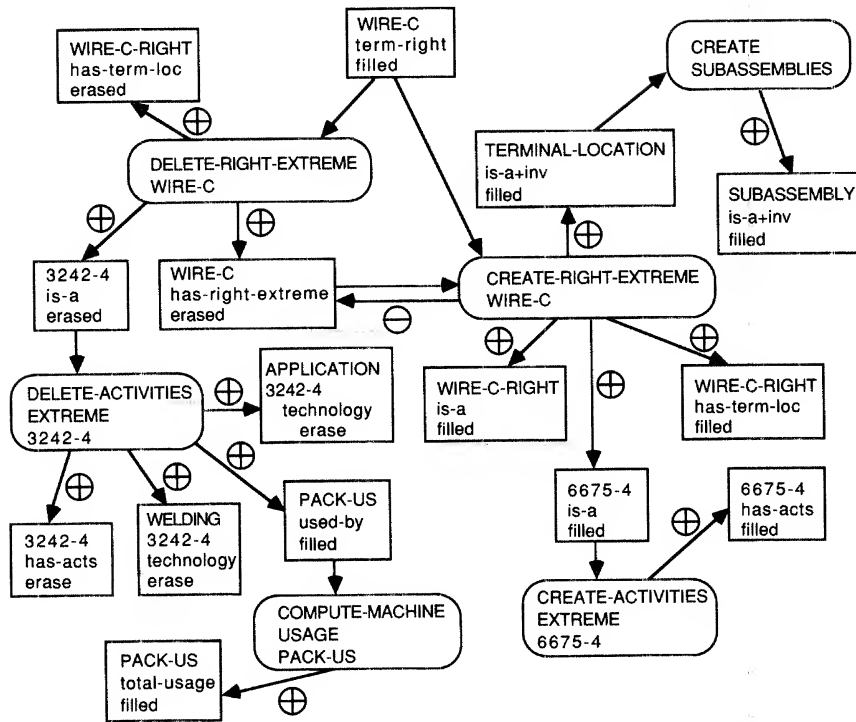


Figure 7-32. First Network of Operators and Conditions for the Example Harness

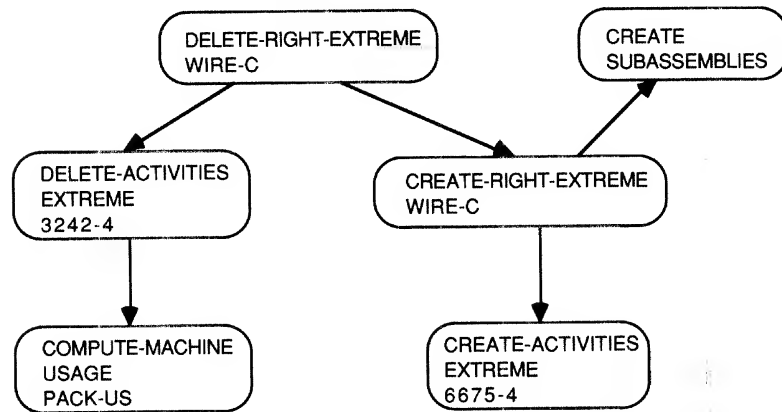


Figure 7-33. First Network of Operators for the Example Harness

1. Operator *Delete-Right-Extreme* deletes the extreme and terminal-location objects associated with the old terminal "3242" before operator *Create-Right-Extreme* creates objects for the new terminal "6675".
2. Operator *Delete-Activities-Extreme* propagates the effect of deleting the object 3242-4 and deletes the associated extreme activities.
3. The *Compute-Machine-Usage* operator updates the usage of those machines related to the activities being deleted.
4. Operator *Create-Subassemblies* updates the set of subassembly schemas after the *Create-Right-Extreme* operator adds a new terminal-location object to the context.
5. The *Create-Activities-Extreme* operator creates the activities associated with the new terminal-location object 6675-4.

After the network of operators is produced by the NIO, control passes to the *Domain Operator Executor* (DOE) and the operators are executed in a sequence that does not violate the precedences. During operator execution, the unpredictable effects of the operators are determined and stored in the agenda. Only two of the six operators have unpredictable effects:

- The *Create-Subassemblies* operator introduces four changes in the agenda: "(sub-1 has-wires filled)", "(wire-a wire-of filled)", "(wire-b wire-of filled)" and "(wire-c wire-of filled)"; and
- The *Create-Activities-Extreme* operator yields the single unpredictable effect "(molding-6675-4 is-a filled)".

In both cases, these changes were not predictable until the DOE invoked the execution of the operators. For example, HARNESS PLANEX did not know which wires would be coupled into which subassemblies.

Figure 7-34 shows the network of operators and conditions created by the FPO after these new changes are added to the agenda. Only the "(molding-6675-4 is-a filled)" change invokes another operator that selects appropriate manufacturing technologies and estimates the duration of the activity objects in the network.

When the NIO is applied to the second network, it produces an operator network consisting of two unlinked operators. HARNESS PLANEX may execute these operators in any order using the DOE. The system estimates the duration of activity *molding-6675-4* is 0 and determines that the appropriate machine is "usm-1". In this process, "(usm-1 used-by filled)" is inserted in the agenda.

The third forward propagation phase is relatively simple (as shown in Figure 7-35). The only change in the agenda, "(usm-1 used-by filled)", activates the *Compute-Machine-Usage* operator and its single effect "(usm-1 total-usage filled)" is inserted in the agenda. This effect does not activate any other operators. An operator network with only one node is produced when the NIO is applied to this small network. The DOE executes the

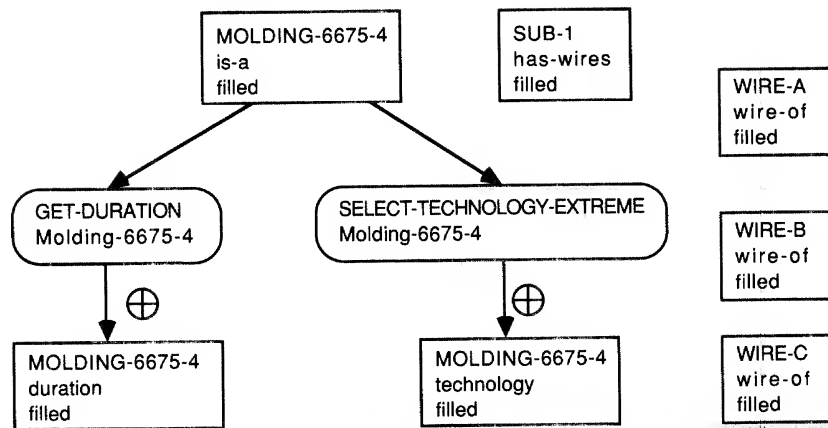


Figure 7-34. Second Network of Operators and Conditions for the Example Harness

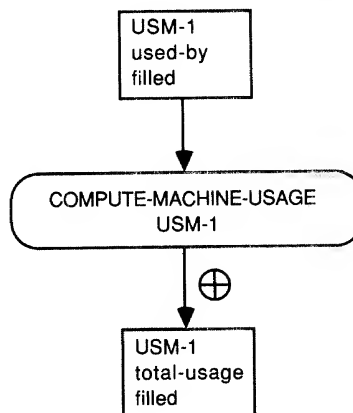


Figure 7-35. Third Network of Operators and Conditions for the Example Harness

operator *Compute-Machine-Usage*, no new unpredictable effects are produced and the propagation of changes is completed.

The results of the planning process for the modified harness are shown in Figures 7-36 to 7-38. Changing the terminal type on the right end of wire "C" reduced the total usage of the machine "pack-us" but increased the usage of the machine "usm-1". Also, wire "C" can be cut using the "cs-26" cutting machine because its ends do not require manual terminal application.

SUB	T	TERM-UM	PEL	CABLE	WIRE	CUT	PEL	TERM-UM	T
1	W	3242	6	952011	A	950.0	10	3197	U T
1	T	3197-U	10	951801	B	640.0	4	6675	X
1	T	3197-U	10	951800	C	1730.0	6	6675	X

Figure 7-36. Updated Process Sheet for the Example Harness

SUB	MOLD-D	MOLD-M	TIN/W	APPL-D	APPL-M	CUT-D	CUT-M	WIRE
	APPL-D	APPL-M	TIN/W	MOLD-D	MOLD-M			
1			0.012	0.126	PACK-US	0.011	CRIMPER	A
	0.144	SC-3197	0.06					
1				0.037	USM-1	0.017	CS-26	B
1						0.025	CS-26	C
			0.012	0.037	USM-1			

Figure 7-37. Updated Time Sheet for the Example Harness

MACHINE	TIME
CRIMPER	0.011
CS-26	0.042
PACK-US	0.126
SC-3197	0.144
TIN	0.060
USM-1	0.074
TOTAL >	0.457

Figure 7-38. Updated Machine Report for the Example Harness

7.4 Conclusions

This chapter illustrated how the components of the PLANEX architecture can be used to develop a system for planning the manufacture of automotive electrical wire harnesses. The basic components of HARNESS PLANEX—representational structures, knowledge sources, domain operators and user interface mechanisms—are instances of the components of PLANEX described in Chapter 4. The behavior of the system includes the three levels of user interaction—strategic, operative and interface—supported by the architecture.

The example showed how the control operators of PLANEX generate and update process plans. The initial plan was produced by executing the domain operators in a predetermined sequence. Modifications to the plan, however, occurred dynamically. Control operators generated strategic meta-plans of operators to propagate the effects of changing the harness design. Such strategic

planning facilitates the operation of HARNESS PLANEX. The user need not be aware of the direct or indirect consequences of modifying planning information and is not responsible for reexecuting those operators whose input has changed.

References

- [1] Arditi, D., "Diffusion of Network Planning in Construction," *Journal of Construction Engineering and Management*, American Society of Civil Engineers (ASCE), Vol. 109, No. 1, pp. 1-12, March 1983.
- [2] Assad, A. A., and Wasil, E. A., "Project Management Using a Microcomputer," *Computers and Operations Research*, Vol. 13, No. 2/3, pp. 231-260, 1986.
- [3] Baker, K. R., *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York, 1974.
- [4] Baracco-Miller, E., *Planning for Construction*, unpublished Master's Thesis, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, May 1987.
- [5] Bell, C., *Use and Justification of Algorithms for Managing Temporal Knowledge in O-Plan*, Technical Report AIAI-TR-6, Artificial Intelligence Applications Institute, University of Edinburgh, 1985.
- [6] Berenji, H. R., and Khoshnevis, B., "Use of Artificial Intelligence in Automated Process Planning," *Computers in Mechanical Engineering (CIME)*, pp. 47-55, September 1986.
- [7] Bremdal, B. A., "Control Issues in a Knowledge-Based Planning System for Ocean Engineering Tasks," *Proceedings, Third International Expert Systems Conference*, London, England, June 2-4, pp. 21-36, 1987.
- [8] Bremdal, B. A., *An Investigation of Marine Installation Processes—A Knowledge-Based Planning Approach*, unpublished Ph.D. Dissertation, Department of Marine Technology, Norwegian Institute of Technology, University of Trondheim, Trondheim, Norway, July 1988.
- [9] Brown, F. M., Ed., *The Frame Problem in Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, 1987.

- [10] Burgess, A. R., and Killebrew, J. B., "Variation in Activity Level on a Cyclical Arrow Diagram," *Journal of Industrial Engineers*, Vol. 13, No. 2, pp. 76-83, February 1962.
 - [11] Chang, T. C., and Wysk, R., *An Introduction to Automated Process Planning Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1985.
 - [12] Chapman, D., *Planning for Conjunctive Goals*, unpublished Master's Thesis, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, November 1985.
 - [13] Charnes, A., Cooper, W. W., and Thompson, G. L., "Critical Path Analysis via Chance Constrained and Stochastic Programming," *Operations Research*, Vol. 12, pp. 460-470, 1964.
 - [14] Christofides, N., *Graph Theory: An Algorithmic Approach*, Academic Press, London, 1975.
 - [15] Corkill, D., "Hierarchical Planning in a Distributed Environment," *Proceedings, Sixth International Joint Conference on Artificial Intelligence*, Tokyo, pp. 168-175, August, 1979.
 - [16] Crowston, W., and Thompson, G. L., "Decision CPM: A Method for Simultaneous Planning, Scheduling and Control of Projects," *Operations Research*, Vol. 15, pp. 407-426, 1967.
 - [17] Crowston, W., "Decision CPM: Network Reduction and Solution," *Journal of the Operational Research Society*, Vol. 21, pp. 435-452, 1970.
 - [18] ———, *MASTERFORMAT—Master List of Section Titles and Numbers*, The Construction Specifications Institute, Alexandria, VA, 1983.
 - [19] Dagostino, F. R., *Estimation in Building Construction*, Reston Publishing Company, New York, 1978.
 - [20] Darwiche, A., Levitt, R. E., and Hayes-Roth, B., "ORPLAN: Generating Project Plans by Reasoning about Objects, Actions and Resources," *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, Vol. 2, No. 3, 1988.
 - [21] Davis E., and Heidorn, G. E., "Optimal Project Scheduling under Multiple Resource Constraints," *Management Science*, Vol. 17, pp. B803-B816, August 1971.
 - [22] Davis E., "CPM Use in Top 400 Construction Firms," *Journal of the Construction Division*, American Society of Civil Engineers (ASCE), Vol. 100, March 1974.
 - [23] Davis E., and Patterson J. H., "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling," *Management Science*, Vol. 21, No. 8, pp. 944-955, April 1975.
-

- [24] de Kleer, J., "An Assumption-Based Truth Maintenance System," *Artificial Intelligence*, Vol. 28, No. 2, pp. 127-162, January 1986.
 - [25] Doyle, J., "A Truth Maintenance System," *Artificial Intelligence*, Vol. 12, No. 3, pp. 231-272, 1979.
 - [26] Elmaghraby, S. E., *Activity Networks: Project Planning and Control by Network Models*, John Wiley and Sons, New York, 1977.
 - [27] Elmaghraby S. E., and Pulat, P. S., "Optimal Project Completion With Due-Dated Events," *Naval Research Logistics Quarterly*, Vol. 26, No. 2, pp. 331-348, June 1979.
 - [28] Erman, L. D., Hayes-Roth, F., Lesser, V. R., and Reddy, D. R., "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys*, Association for Computing Machinery (ACM), Vol. 12, pp. 213-253, February 1980.
 - [29] Ernst, G. W., and Newell, A., *GPS: A Case Study in Generality and Problem Solving*, Academic Press, New York, 1969.
 - [30] Falk, J. E., and Horowitz, J. L., "Critical Path Problems with Concave Cost-Time Curves," *Management Science*, Vol. 19, No. 4, pp. 446-455, 1972.
 - [31] Fennes, S. J., "Tabular Decision Logic for Structural Design," *Journal of the Structural Division*, American Society of Civil Engineers (ASCE), Vol. 92, No. ST6, pp. 473-490, June 1966.
 - [32] Fennes, S. J., Flemming, U., Hendrickson, C., Maher, M. L., and Schmitt, G., "An Integrated Software Environment for Building Design and Construction," *Proceedings, Fifth Conference on Computers in Civil Engineering*, American Society of Civil Engineers (ASCE), 1988.
 - [33] Fikes, R. E., and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, No. 4, pp. 189-208, 1971.
 - [34] Filman, R. E., "Reasoning with Worlds and Truth Maintenance in a Knowledge-Based System Shell," *Communications of the Association for Computing Machinery (CACM)*, Association for Computing Machinery (ACM), Vol. 31, No. 4, pp. 382-401, April 1988.
 - [35] Fondahl, J. W., *A Non-Computer Approach to the Critical Path Method for the Construction Industry*, Technical Report 9, Construction Institute, Stanford University, Palo Alto, CA, 1962.
 - [36] Fulkerson, D. R., "A Network Flow Computation for Project Cost Curves," *Management Science*, Vol. 7, pp. 167-178, 1961.
-

- [37] Garman, M. B., *Solving Combinatorial Decision Problems with Interactive Computer Graphics, with Application to Job-Shop Scheduling*, unpublished Ph.D. Dissertation, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA, May 1970.
 - [38] Garrett, J. H., Jr., and Fenves, S. J., *SPEX: A Knowledge-Based Standard Processor for Structural Component Design*, Technical Report R-86-157, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, September 1986.
 - [39] Glover, F., Hultz, J., and Klingman, D., "Improved Computer-Based Planning Techniques, Part I," *Interfaces*, Vol. 8, No. 4, pp. 16-24, August 1978.
 - [40] Glover, F., Hultz, J., and Klingman, D., "Improved Computer-Based Planning Techniques, Part II," *Interfaces*, Vol. 9, No. 4, pp. 12-20, August 1979.
 - [41] Gupta, S. K., and Taube, L. R., "A State of the Art Survey of Research on Project Management," in *Project Management: Methods and Studies*, Vol. 11, *Studies in Management Science and Systems*, Elsevier Science Publishers B.V. (North-Holland), 1985.
 - [42] Halpin, D. W., and Woodhead, R. W., *Design of Construction and Process Operations*, John Wiley and Sons, New York, 1976.
 - [43] Halpin, D. W., "CYCLONE—Method for Modeling Job Site Processes," *Journal of the Construction Division*, American Society of Civil Engineers (ASCE), Vol. 103, No. CO3, pp. 489-499, September 1977.
 - [44] Halpin, D. W., Escalona, A. L., and Szmurlo, P. M., *Work Packaging for Project Control. A Report to the Construction Industry Institute*, Technical Report, University of Maryland, August 1987.
 - [45] Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence*, Vol. 26, No. 3, pp. 251-321, 1985.
 - [46] Henderson, M. R., and Anderson, D. C., "Computer Recognition and Extraction of Form Features: A CAD/CAM Link," *Computers in Industry*, Vol. 5, pp. 329-339, June 1984.
 - [47] Hendrickson, C., and Janson, B. N., "A Common Network Flow Formulation for Several Civil Engineering Problems," *Civil Engineering Systems*, Vol. 1, pp. 195-203, June 1984.
 - [48] Hendrickson, C. T., Martinelli, D., and Rehak, D. R., "Hierarchical Rule-Based Activity Duration Estimation," *Journal of Construction Engineering and Management*, American Society of Civil Engineers (ASCE), Vol. 113, No. 2, pp. 288-301, June 1987.
-

- [49] Hendrickson, C., Zozaya-Gorostiza, C. A., Rehak, D. R., Baracco-Miller, E. G., and Lim, P. S., "An Expert System for Construction Planning," *Journal of Computing in Civil Engineering*, American Society of Civil Engineers (ASCE), Vol. 1, No. 4, pp. 253-269, October 1987.
 - [50] Hendrickson, C., and Au, T., *Project Management for Construction: Fundamental Concepts for Owners, Engineers, Architects and Builders*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
 - [51] Hendrickson, C., and Zozaya-Gorostiza, C. A., "A Unified Activity Network Model," *Journal of Computing in Civil Engineering*, American Society of Civil Engineers (ASCE), Vol. 3, No. 2, April 1989.
 - [52] Hindelang, T. J., and Muth, J. F., "A Dynamic Programming Algorithm for Decision CPM Networks," *Operations Research*, Vol. 27, No. 2, pp. 225-241, March-April 1979.
 - [53] Hurley, R. B., *Decision Tables in Software Engineering*, Van Nostrand Reinhold, New York, 1983.
 - [54] Husbands, P., Mill, F., and Warrington, S., "A Knowledge Based Process Planning System," *Knowledge Based Expert Systems in Engineering: Planning and Design*, Sriram, D., and Adey, R. A., Eds., Second International Conference on AI in Engineering, Boston, MA, Computational Mechanics Publications, pp. 439-448, 1987.
 - [55] Ibramsha, M., and Rajaraman, U., "Detection of Logical Errors in Decision Table Programs," *Communications of the Association for Computing Machinery (CACM)*, Association for Computing Machinery (ACM), Vol. 21, No. 12, pp. 1016-1024, December 1978.
 - [56] Johnson, H., and Kaplan, R. S., *Relevance Lost, The Rise and Fall of Management Accounting*, Harvard Business School Press, Boston, MA, 1987.
 - [57] Kelley, J. E., Jr., "Critical Path Planning and Scheduling: Mathematical Basis," *Operations Research*, Vol. 9, No. 3, pp. 296-320, 1961.
 - [58] Kennington, J. L., and Helgason, R. V., *Algorithms for Network Programming*, John Wiley and Sons, New York, 1980.
 - [59] King, P. J. H., and Johnson, R. G., "The Conversion of Decision Tables to Sequential Testing Procedures," *Computer Journal*, Vol. 18, pp. 298-306, 1975.
 - [60] Kurtulus, I., and Davis, E. W., "Multi-Project Scheduling: Categorization of Heuristic Rules Performance," *Management Science*, Vol. 28, No. 2, pp. 161-172, February 1982.
 - [61] Lawler, E., *Combinatorial Optimization*, Holt-Rinehart and Winston, New York, 1976.
-

- [62] Levitt, R. E., and Kunz, J. C., "Using Knowledge of Construction and Project Management for Automated Schedule Updating," *Project Management Journal*, Vol. XVI, No. 5, pp. 57-76, December 1985.
- [63] Levitt, R. E., "Expert Systems in Construction: State of the Art," in *Expert Systems for Civil Engineering: Technology and Applications*, Maher, M. L., Ed., American Society of Civil Engineers (ASCE), New York, Ch. 6, pp. 85-112, 1987.
- [64] Levy, F. K., Thompson, G. L., and Wiest, J. D., "Multi-Ship, Multi-Shop, Workload Smoothing Program," *Naval Research Logistics Quarterly*, Vol. 9, No. 1, pp. 37-44, 1962.
- [65] Lim, P. S., and Hendrickson, C., *Description of the Prototype EXCAVATION PLANEX System*, Technical Report, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, September 1987.
- [66] Malcolm, D. G., Roseboom, J. H., Clark, C. E., and Fazar, W., "Applications of a Technique for Research and Development Program Evaluation," *Operations Research*, Vol. 7, No. 5, pp. 646-669, 1959.
- [67] Marshall, G., Barber, T. J., and Boardman, J. T., "Methodology for Modelling a Project Management Control Environment," *IEE Proceedings*, Vol. 134, No. 4, pp. 287-300, July 1987.
- [68] McCarthy, J., and Hayes, P., "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 4*, Meltzer, B., and Michie, D., Eds., Edinburgh University Press, Edinburgh, UK, pp. 463-502, 1969.
- [69] McGartland, M. R., and Hendrickson, C., "Expert Systems for Construction Project Monitoring," *Journal of Construction Engineering and Management*, American Society of Civil Engineers (ASCE), Vol. 111, No. 3, pp. 293-307, September 1985.
- [70] Mahoney, W. D., and Thornley, A., Eds., *Means Square Foot Costs*, Fifth Edition, R. S. Means Company, Inc., 1984.
- [71] Mahoney, W. D., and Thornley, A., Eds., *Building Construction Cost Data*, 45th Edition, R. S. Means Company, Inc., 1987.
- [72] Moder, J. J., Phillips, C. R., and Davis, E. W., *Project Management with CPM, PERT and Precedence Diagramming*, Third Edition, Van Nostrand Reinhold Co., New York, 1983.
- [73] Moder, J. J., and Crandall, K. C., "Precedence Diagramming: Time Computations, Anomalies and Remedies," in *Project Management: Methods and Studies*, Vol. 11, *Studies in Management Science and Systems*, Elsevier Science Publishers B.V. (North-Holland), 1985.

- [74] Mouleeswaran, C. B., and Fischer, H. G., "A Knowledge Based Environment for Process Planning," *Applications of Artificial Intelligence in Engineering Problems*, Vol. 2, Sriram, D., and Adey, R. A., Eds., First International Conference on AI in Engineering, Southampton, England, Springer-Verlag, Berlin, pp. 1013-1027, April, 1987.
- [75] Navichandra, D., Sriram, D., and Logcher, R. D., "GHOST: Project Network Generator," *Journal of Computing in Civil Engineering*, American Society of Civil Engineers (ASCE), Vol. 2, No. 3, pp. 239-254, July 1988.
- [76] Nunnally, S. W., *Construction Methods and Management*, Second Edition, Prentice-Hall, Englewood Cliffs, NJ, 1987.
- [77] Opitz, H., *A Classification System to Describe Workpieces*, Pergamon Press, Elmsford, NY, 1970.
- [78] Patterson, J. H., and Groth, R., "Scheduling a Project under Multiple Resource Constraints: A Zero-One Programming Approach," *AIIE Transactions*, Vol. 8, No. 3, pp. 449-456, December 1976.
- [79] Patterson J. H., "A Comparison on Exact Approaches for Solving Multiple Constrained Resource Project Scheduling Problem," *Management Science*, Vol. 30, No. 7, pp. 854-867, July 1984.
- [80] Pearl, J., *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison Wesley, Reading, MA, 1984.
- [81] Petrovic, R., "Optimization of Resource Allocation in Project Planning," *Operations Research*, Vol. 16, pp. 559-567, May-June 1968.
- [82] Pollack, S. L., Hicks, H. T., and Harrison, W. T., *Decision Tables: Theory and Practice*, The Wiley Communigraph Series on Business Data Processing, 1971.
- [83] Romero, H., *Strategic Planning and Monitoring System for a Robotic Excavator*, unpublished Master's Thesis, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, September 1988.
- [84] Sacerdoti, E. D., "Planning in a Hierarchy of Abstraction Spaces," *Proceedings, Third International Joint Conference on Artificial Intelligence*, Stanford, CA, pp. 412-422, 1973.
- [85] Sacerdoti, E. D., "The Nonlinear Nature of Plans," *Proceedings, Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, USSR, pp. 206-214, September, 1975.
- [86] Sacerdoti, E. D., *A Structure for Plans and Behavior*, Elsevier-Holland, New York, 1977.

- [87] Sathi, A., and Fox, M. S., "Representation of Activity Knowledge for Project Management," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Institute for Electrical and Electronics Engineers (IEEE), Vol. PAMI-7, No. 5, pp. 531-552, September 1985.
 - [88] Sathi, A., Morton, E., and Roth, S. F., "Callisto: An Intelligent Project Management System," *AI Magazine*, Vol. 7, No. 5, pp. 34-52, Winter 1986.
 - [89] Schmitt, G., *ARCHPLAN—An Architectural Front End to Engineering Design Expert Systems*, Technical Report, Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, September 1987.
 - [90] Smith, L. A., and Mandakovic, T., "Estimating: The Input into Good Project Planning," *IEEE Transactions on Engineering Management*, Institute for Electrical and Electronics Engineers (IEEE), Vol. EM32, No. 4, pp. 181-185, 1985.
 - [91] Sriram, D., and Adey, R. A., Eds., *Knowledge-Based Expert Systems in Engineering: Planning and Design*, Computational Mechanics Publications, 1987.
 - [92] Stallman, R. M., and Sussman, G. J., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, Vol. 9, No. 2, pp. 135-196, October 1977.
 - [93] Stefik, M., *Planning with Constraints*, unpublished Ph.D. Dissertation, Department of Computer Science, Stanford University, Palo Alto, CA, January 1980.
 - [94] Stefik, M., "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, Vol. 16, No. 2, pp. 111-140, 1981.
 - [95] Stefik, M., "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence*, Vol. 16, No. 2, pp. 141-170, 1981.
 - [96] Stinson, J. P., Davis, E. W., and Khumawala, M., "Multiple-Resource Constrained-Scheduling Using Branch and Bound," *AIIE Transactions*, Vol. 10, No. 3, pp. 252-259, 1978.
 - [97] Talbot, F. B., and Patterson, J. H., "An Efficient Integer Programming Algorithm With Network Cuts for Solving Resource-Constrained Scheduling Problems," *Management Science*, Vol. 24, No. 12, pp. 1163-1174, 1978.
 - [98] Talbot, F. B., "Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case," *Management Science*, Vol. 28, No. 10, pp. 1197-1210, 1982.
 - [99] Tate, A., *INTERPLAN: A Plan Generation System Which Can Deal With Interactions Between Goals*, Technical Report MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh, 1974.
 - [100] Tate, A., *Using Goal Structure to Direct Search in a Problem Solver*, unpublished Ph.D. Dissertation, Machine Intelligence Research Unit, University of Edinburgh, 1975.
-

- [101] Tate, A., "Generating Project Networks," *Proceedings, Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, pp. 888-893, 1977.
- [102] Tate, A., *A Review of Knowledge-Based Planning Techniques*, Technical Report AIAI-TR-9, Artificial Intelligence Applications Institute, University of Edinburgh, 1985.
- [103] Thompson, G. L., "CPM and DCPM Under Risk," *Naval Research Logistics Quarterly*, Vol. 15, No. 2, pp. 233-240, June 1968.
- [104] Vere, S. A., "Planning in Time: Windows and Durations for Activities and Goals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Institute for Electrical and Electronics Engineers (IEEE), Vol. PAMI-5, No. 5, pp. 246-259, May 1983.
- [105] Spradlin, W. H., Jr., Ed., *Walker's Building Estimator's Reference Book*, 22nd Edition, Frank R. Walker Company, 1986.
- [106] Wang, H-P., "A Knowledge-Based Computer-Aided Process Planning System," *Knowledge Based Expert Systems in Engineering: Planning and Design*, Sriram, D., and Adey, R. A., Eds., Second International Conference on AI in Engineering, Boston, MA, Computational Mechanics Publications, pp. 259-272, 1987.
- [107] Waterman, D. A., *A Guide to Expert Systems*, Addison Wesley, Reading, MA, 1986.
- [108] Welland, R., *Decision Tables and Computer Programming*, Heyden, London, 1981.
- [109] Wiest, J. D., "A Heuristic Model for Scheduling Large Projects with Limited Resources," *Management Science*, Vol. 13, No. 6, pp. B359-B377, 1967.
- [110] Wiest, J. D., and Levy, F. K., *A Management Guide to PERT/CPM: with GERT/PDM/DCPM and Other Networks*, Second Edition, Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [111] Wiest, J. D., "Gene-Splicing PERT and CPM: The Engineering of Project Network Models," in *Project Management: Methods and Studies*, Vol. 11, *Studies in Management Science and Systems*, Elsevier Science Publishers B.V. (North-Holland), 1985.
- [112] Wilkins, D. E., *Practical Planning*, Morgan Kaufmann Publishers, San Mateo, CA, 1988.
- [113] Willis, R. J., and Hastings N. A. J., "Project Scheduling with Resource Constraints Using Branch and Bound Methods," *Journal of the Operational Research Society*, Vol. 27, No. 2, pp. 341-349, 1976.
- [114] Zozaya-Gorostiza, C. A., and Hendrickson, C., "An Expert System for Traffic Signal Setting Assistance," *Journal of Transportation Engineering*, American Society of Civil Engineers (ASCE), Vol. 13, No. 2, March 1987.

- [115] Zozaya-Gorostiza, C. A., *Knowledge-Based Planning for Construction Projects*, unpublished Ph.D. Dissertation, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, May 1988.
 - [116] Zozaya-Gorostiza, C. A., and Lim, P. S., *CONSTRUCTION PLANEX User Manual*, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA, 1988.
-

Index

- Abstraction, 20, 138, 142
- ABSTRIPS, 10, 20, 82, 142
- Actions, 92, 126
- Activities, 27, 66, 256, 259, 268, 272
 - See also* Element activities, Project activities
- Activity durations. *See* Durations
- Activity formulation, 1, 72, 148, 151, 179
- Activity network, 2, 38, 45, 73, 126, 225
- Activity operators, 268
- Activity-centered planners, 4
- Activity-on-branch diagram, 38
- Activity-on-node diagram, 38, 45, 225
- Agenda, 9, 104, 108, 129, 146, 164, 166, 246
- AI. *See* Artificial Intelligence
- AI-based planners, 14
- Analysis, 62, 81, 173
- Animation, 11, 134, 227
- ANIMATOR, 12, 134, 222, 227
- AOB. *See* Activity-on-branch diagram
- AON. *See* Activity-on-node diagram
- Application object, 89
- Applications, 148, 154
- Architecture, 100, 104, 138, 148, 158, 164, 185, 258
- ARCHPLAN, 234
- Artificial intelligence, 6, 14
- Automated planning, 232
- Backtracking, 17, 34
- Backward planning, 103
- Backward Search Operator, 10, 109, 113, 118, 129, 142, 166, 174, 230
- Backward Search Operator algorithm, 116
- Basic CPM. *See* CPM
- Basic Critical Path Method. *See* CPM
- Behavior, 229, 276
- Binding, 97
- Blackboard, 32
- Blackboard model, 68
- Blackboard planners, 32, 69
- Blackboard planning, 32, 66, 69
- BLOCKS PLANEX, 137, 163
- BLOCKS PLANEX architecture, 164
- BLOCKS PLANEX examples, 166, 170
- Blocks world, 17, 24, 137, 163
- Blocks-world planning, 163
- Body operators, 264
- Bottom-up activity formulation model, 72, 148, 151, 179, 248
- Branch-and-bound method, 40, 46
- BSO. *See* Backward Search Operator
- CAD. *See* Computer-Aided Design
- CALLISTO, 61
- CAM. *See* Computer-Aided Manufacturing

- Chapman, D., 23
 Chez Dan, 1
 CIC. *See* Computer-Integrated Construction
 Coding, 50, 52
 Computer-Aided Design, 5
 Computer-Aided Manufacturing, 5
 Computer-Integrated Construction, 5, 178
 Conceptualization of plans, 148
 Conditions, 92, 126
 Constraint posting, 30
 Constraints, 30
 Construction methods, 180
 CONSTRUCTION PLANEX, 11, 155, 177
 CONSTRUCTION PLANEX architecture, 185
 CONSTRUCTION PLANEX behavior, 229
 CONSTRUCTION PLANEX evaluation, 244
 CONSTRUCTION PLANEX examples, 234, 236
 CONSTRUCTION PLANEX use, 229
 Construction planning, 48, 149, 154, 155, 177, 178, 237, 245
 Context, 88
 Contingency management, 2
 See also Scheduling
 Control, 70, 82, 100, 101, 104, 108, 141, 174
 Control behavior, 101
 Control operators, 88, 104, 109, 146, 164
 CONTROL PANEL, 10, 91, 125, 129, 222, 230
 Control problem, 32
 Cooking, 1
 Costs, 3, 58, 181, 209, 257
 See also Estimation
 CP. *See* CONTROL PANEL
 CPM, 36, 38, 40, 60, 71, 74
 Crews, 180
 See also Technology choice
 Critical Path Method. *See* CPM
 Criticality number, 20
 Critics, 23
 Crowston, W., 44
 Current object, 89
 CYCLONE, 57
 Database management system, 251
 DCPM. *See* Decision CPM
 Decision CPM, 44, 57, 76, 181
 Decision table, 91, 92, 125
 Design, 5
 Design element operators, 195
 Design elements, 49, 50, 155, 179, 185, 188, 195, 196
 See also Design features
 Design features, 49, 50, 72
 Design of planning systems, 150
 Deterministic scheduling, 36
 DEVISER, 26
 Dijkstra, E. W., 38
 DOE. *See* Domain Operator Executor
 Domain Operator Executor, 10, 109, 121, 230, 246, 286
 Domain Operator Executor algorithm, 122
 Domain Operator Schema, 9, 104, 105, 110, 125, 142, 164, 174, 195, 259
 Domain operators, 9, 88, 105, 142, 146, 152, 157, 159, 162, 195, 259
 DOS. *See* Domain Operator Schema
 Durations, 2, 3, 38, 58, 181, 202, 206, 218, 257, 270, 275
 See also Estimation
 Dynamic programming, 40
 Element activities, 155, 179, 185, 191, 196, 198, 211, 256
 Element activity operators, 198
 Ernst, G. W., 16
 Estimation, 49, 58, 72, 181, 257
 Evaluation, 170, 244
 Events, 27, 38, 74
 Examples, 166, 170, 234, 236, 280
 EXCAVATION PLANEX, 11, 137, 158
 EXCAVATION PLANEX architecture, 158
 EXCAVATION PLANEX use, 161
 Excavation planning, 149, 158
 Execution phase, 101
 Expert system, 6
 Explanations, 84, 133, 175, 223, 228, 276
 EXPLORER, 11

- External factors, 2
- Extreme operators, 265
- Fast-track schedule, 183, 210, 238
- Fikes, R. E., 16
- Firing mechanism, 97, 126, 141
- Float, 38, 76
- Floyd-Warshall algorithm, 78, 210
- FOOTER, 234
- Forward pass, 222, 232
- Forward planning, 103
- Forward Propagation Operator, 10, 109, 118, 129, 147, 174, 230, 246, 284
- Forward Propagation Operator algorithm, 112
- FPO. *See* Forward Propagation Operator
- Frame, 9
- Frame problem, 34
- Frank-Wolfe algorithm, 40
- GANTT Interactive Scheduler, 11, 126, 221, 225
- General Problem Solver, 16, 30, 83
- Generalized networks, 47
- Generative planners, 3, 8
- GHOST, 53
- Glover, F., 47, 48
- GPS. *See* General Problem Solver
- Graphics, 84, 175, 225
- HARNESS PLANEX, 11, 137, 161, 253
- HARNESS PLANEX architecture, 258
- HARNESS PLANEX behavior, 276
- HARNESS PLANEX examples, 280
- HARNESS PLANEX use, 276
- Hayes-Roth, B., 32, 82
- HEARSAY-II, 32
- Hierarchical decomposition, 68
- Hierarchical planning, 142
- Hindelang, T. J., 46
- Hole, 150, 158
- Hultz, J., 47, 48
- Hybrid model, 69, 229, 276
- IBDE. *See* INTEGRATED BUILDING DESIGN ENVIRONMENT
- IG. *See* INPUT GENERATOR
- Implementation of planning systems, 153
- INPUT GENERATOR, 12, 222, 234
- Input object, 89, 105
- INTEGRATED BUILDING DESIGN ENVIRONMENT, 12, 234, 232
- Interactive graphics, 126
- See also* Graphics
- INTERPLAN, 20, 25, 70
- Job shop scheduling, 60
- Klingman, D., 47, 48
- Knowledge abstraction, 138
- Knowledge acquisition, 94, 125, 126, 175, 178, 254, 276
- Knowledge base, 88, 156, 159, 162, 210, 271
- KNOWLEDGE CRAFT, 11
- Knowledge hierarchies, 81, 139, 152
- Knowledge representation, 80, 91, 138, 172
- KNOWLEDGE SOURCE ACQUISITION MODULE, 11, 90, 125, 172
- Knowledge Source Activation Record, 32, 104
- Knowledge source evaluation, 96
- KNOWLEDGE SOURCE EVALUATOR, 9, 89, 96, 125, 138, 156, 172
- KNOWLEDGE SOURCE EVALUATOR algorithm, 98
- Knowledge source implementation, 93
- Knowledge Source Schema Definition, 94
- Knowledge sources, 9, 32, 88, 91, 93, 138, 156, 159, 162, 172, 185, 210, 271
- Knowledge-based system, 6
- KS. *See* Knowledge source
- KSAM. *See* KNOWLEDGE SOURCE ACQUISITION MODULE

- KSAR. *See* Knowledge Source Activation Record
- KSE. *See* KNOWLEDGE SOURCE EVALUATOR
- Lag, 2, 29, 207, 220
- Lead, 75, 207, 220
- Levitt, R. E., 7, 60
- LIFT-2, 54
- LIFT-3, 55
- Line balancing, 56
- Linear planners, 15
- Linear programming, 38, 40
- LISP, 11
- Literals, 17, 27
- Longest path, 38
- LP. *See* Linear Programming
- Machine operators, 270
- Machines, 259, 266, 270
- Manufacturing methods, 257
- Manufacturing planning, 48, 149, 161, 253, 280, 284
- MASON, 58, 182
- MASTERFORMAT, 52, 56, 150, 185
- Material package, 201, 209, 214
- Materials, 201, 214
- Meal planning, 1
- Means-ends analysis, 16, 30
- Means-ends model, 67
- Means-ends planning, 66, 67
- Menus, 104, 132, 221, 276
- Meta-operator, 146
- Meta-planners, 29
- Meta-planning, 29, 146
- Methods, 55
 See also Construction methods,
 Manufacturing methods
- MOLGEN, 9, 30, 35, 55
- Muth, J. F., 46
- Net present value, 210
- NETFORM, 47
- Network Interpretation Operator, 10, 109, 118, 166, 230, 246, 284
- Network Interpretation Operator
 algorithm, 120
- Newell, A., 16
- Nilsson, N. J., 16
- NIO. *See* Network Interpretation Operator
- NOAH, 14, 23, 26, 34, 54, 65, 83, 101, 145
- NONLIN, 25, 26, 65, 67, 70
- Nonlinear planners, 23
- Nonlinear planning, 23, 144, 248
- NPV. *See* Net present value
- OARPLAN, 54
- Object-centered planners, 4
- Objects, 9, 88, 89, 93
- Operative planning, 70, 229, 276
- Operator effects, 90
- Operator feasibility, 83, 114
- Operator hierarchies, 82, 173
- Operator network, 103, 168, 170, 284, 286
- Operator precedences, 118
- Operators, 9, 81, 93, 105, 157, 159, 173
 See also Control operators, Domain operators
- Opitz, 50
- OPM, 32, 55, 68, 174
- Opportunistic planning, 146
- Optimization, 35
- Output object, 89, 105
- Output reports, 84, 130, 175, 227, 240, 278, 283, 288
- Passive output graphics, 133
 See also Graphics
- PERT, 36, 141, 182
- PIPPA, 54
- Plan formulation, 5, 14
- PLANEX, 8, 10, 137
- PLANEX applications, 138, 148, 154
- PLANEX architecture, 88, 100, 138, 148
- PLANEX control, 101, 141
- PLANEX development, 10
- PLANEX evaluation, 170

- PLANEX operators, 89
- PLANEX problem solving, 141
- PLANEX requirements, 79
- PLANEX use, 10, 148
- Planning models, 44, 149
- Planning phase, 101, 103, 118
- Planning tools, 5
- PLATFORM, 60
- Precedence diagramming, 38
- Precedence relationships, 50, 179
- Precedences, 2, 74, 179, 183, 207, 219
 - See also* Activity network
- Primitive operators, 23
- Probabilistic scheduling, 36
- Problem solving, 100, 141
- Problem-solving operators, 81, 173, 185, 195, 224, 259
 - See also* Domain operators
- Procedural net, 23
- Process planning, 1, 3, 4, 13, 66, 71, 79, 88, 100, 137, 148
- Process sheet, 253, 276, 278, 283, 288
- Product components, 66
 - See also* Design elements
- Project activities, 1, 155, 179, 185, 193, 201, 203, 204, 216, 256
- Project activity operators, 203
- Project Evaluation and Review Technique. *See* PERT
- Project management, 2
 - See also* Scheduling
- Project network. *See* Activity network
- Project scheduling. *See* Scheduling
- PROPLAN, 59
- Quantity take-offs. *See* Work quantities
- Questions, 133
- Relations, 88
- Report Format Schema, 131
- REPORT GENERATOR, 11, 91, 130, 175, 223, 227, 240
- Reports. *See* Output reports
- Representational structures, 9, 88, 152, 155, 158, 162, 185, 258
- Requirements, 79
- Resource allocation, 41
- Resource leveling, 41
- Resources, 1, 3, 41, 66
- Retrieval-based planners, 3
- RG. *See* REPORT GENERATOR
- Rules, 92, 126
- Sacerdoti, E. D., 20, 23
- Scheduling, 2, 3, 13, 35, 41, 44, 49, 60, 72, 73, 74, 126, 127, 183, 210
- Schema, 9
- Search space, 14
- Shortest path, 38, 47
- Simulation, 57, 134
- SIPE, 35
- Slack, 38, 76
- Slots, 88, 93
- Solution space, 14, 15
- SPAR-1, 43
- SPEX, 92, 234
- STANLEY, 234
- State tree, 16
- Stefik, M., 29
- Strategic planning, 70, 101, 229, 276
- STRIPS, 16, 34
- STRYPES, 234
- Subassemblies, 254, 256, 262
- Successors. *See* Precedences
- Synthesis, 62, 81, 173
- Tate, A., 14
- Technology choice, 2, 3, 49, 55, 72, 180, 204, 217, 257, 273
- Thompson, G. L., 44
- Time constraints, 26
- Time-cost trade-offs, 38, 40, 76, 180
- TIPPS, 55
- TRALI, 92
- Truth maintenance system, 34
- TWEAK, 113

- Unified activity network model, 73, 183, 248
 - Units, 200, 214
 - User assistant, 8, 123, 232
 - User interaction, 84, 88, 90, 123, 132, 158, 161, 163, 175
 - User interface, 220, 276
 - Values, 88, 93
 - Vere, S. A., 26
 - Waterman, D. A., 84, 123
 - Window, 2, 28
 - Wire bodies, 256, 259, 260, 264, 268
 - Wire extremes, 256, 259, 261, 262, 265, 266, 269, 267
 - Wire operators, 260
 - Wires. *See* Wire bodies, Wire extremes
 - Work breakdown, 50
 - Work quantities, 199, 205, 213, 214
 - Work tasks, 3, 49, 50, 179, 255
 - Work-centered planners, 4
-

Carlos Zozaya-Gorostiza is a Systems Researcher in the firm Condumex. He obtained his B.S. in Chemical Engineering from the Universidad Autonoma Metropolitana in Mexico City and M.S. and Ph.D. degrees in Civil Engineering from Carnegie Mellon University in 1985 and 1988, respectively. He has also worked as a Computer Systems Manager for Macopel and as a Project Engineer for Asimex in Mexico. He continues to develop computer aids for process planning and logistics decision making for several industrial clients in Mexico.

Chris Hendrickson is a Professor of Civil Engineering and an affiliate faculty member of the Engineering Design Research Center at Carnegie Mellon University. He has co-authored two textbooks, *Transportation Investment and Pricing Principles* (John Wiley & Sons, 1984) and *Project Management for Construction* (Prentice-Hall, 1989), and has published numerous articles in the professional literature. His education includes B.S. and M.S. degrees from Stanford University, a B.Phil. degree in economics from Oxford University, and a Ph.D. from the Massachusetts Institute of Technology. Prof. Hendrickson has received a Rhodes Scholarship and the Benjamin Teare Teaching Award.

Daniel R. Rehak is an Associate Professor of Civil Engineering at Carnegie Mellon University. He also is an affiliate faculty member of the Engineering Design Research Center and the Robotics Institute. He obtained his B.S. and M.S. degrees in Civil Engineering from Carnegie-Mellon in 1973 and 1975, respectively, and his Ph.D. in Civil Engineering from the University of Illinois at Urbana-Champaign in 1981. He joined Carnegie-Mellon University in 1981, and is the author of over 75 technical papers and reports in the area of advanced computer applications in Civil Engineering. Prof. Rehak received a 1985 NSF Presidential Young Investigator Award.
